

DSA – Formação Cientista de Dados

9. Visualização de Dados e Design de Dashboards

9.1. Introdução

O Que É Visualização de Dados?

Chamamos de Visualização de Dados, Data Visualization ou “DataViz”, o estudo e a criação de representações visuais para dados. Em outras palavras, fazer gráficos!

Um termo interessante que tem ganhado destaque nos últimos anos é “Visual Literacy”, que é a capacidade de expor e interpretar dados de forma visual.

Ou seja, não só fazer gráficos, e sim saber escolher os gráficos mais adequados para cada tipo de informação e entender a mensagem que os dados querem transmitir.

Encontrar os insights certos, que possam fazer a diferença na tomada de decisão, não significa nada se você não sabe como comunicar suas descobertas para os outros.

Para um bom projeto de visualização, você deve conhecer e compreender verdadeiramente seu público-alvo e a finalidade para a qual você está projetando.

Benefícios da Visualização de Dados

- Absorção de Informação
- Visualização de Padrões de Negócios
- Identificação de Tendências
- Manipulação e interpretação de dados
- Conversão em linguagem de negócio

Compreendendo Sua Audiência

- Tipo de audiência
- Faça os dados contarem a história
- Use estudos de caso
- Público não técnico

Entregue os resultados de forma ética.

O Que É Storytelling?

É a arte de contar histórias envolventes. É a capacidade de contar histórias de maneira relevante, onde os recursos audiovisuais são utilizados junto com as palavras.

Ao usar storytelling, as empresas podem:

- Garantir que a marca será diretamente conectada à mensagem
- Fazer com que os consumidores entendam que a empresa vai entregar produtos de qualidade
- Se destacar da concorrência

O intuito de contar uma história é encantar e cativar uma audiência.

Storytelling é a mais antiga forma de passar conhecimento através de gerações.

Na prática, o ser humano pratica análise de dados desde os primeiros anos de vida. E ser capaz de visualizar os dados e contar a história que eles querem dizer é uma habilidade cada vez mais valorizada. Visualizar os dados e contar a história é a chave que pode levar a decisões cada vez mais precisas.

Os dados possuem uma história e as ferramentas tecnológicas não sabem como contar a história, pelo menos ainda. Exatamente aí que o Cientista de Dados precisa ser um contador de histórias. Precisa ser mais que um analista, precisa ser um comunicador, capaz de dar vida a história e contextualizá-la no mundo real.

Uma visualização efetiva de dados pode ser a diferença entre sucesso e falha nas decisões de negócio.

Técnicas de design começam a ser incorporadas ao mundo de Data Science, ou seja, pensar o design do produto final (a história) de forma que a comunicação seja a mais efetiva possível.

O processo de comunicação dos dados é realizado levando em consideração como o usuário que ouvirá a história irá interpretá-la.

Um contador de histórias de dados descobre o ponto de vista mais convincente para orientar seus usuários a tomar uma decisão lógica, que é suportada com provas e não apenas com opinião de especialistas.

Propósito da Visualização de Dados

Visualização de dados é importante para comunicar a mensagem de forma rápida e eficiente.

Pirâmide do aprendizado: 90% Fazer, 70% Falar, 50% Olhar e Ouvir, 30% Olhar ou Assistir, 20% Ouvir e 10% Ler.

As visualizações devem mostrar o contexto, ou seja, o sistema ao qual pertencem as informações ou as correlações entre os dados.

Com o fenômeno do Big Data, volumes cada vez maiores de dados estão disponíveis para empresas, que precisam tornar viável a leitura desses dados, através de visualizações adequadas.

A demanda por ferramentas fáceis de usar e que tenham uma boa relação custo-benefício ainda é uma das dificuldades desse mercado.

Processos e habilidades necessários para lidar com as crescentes demandas de visualização de dados:

- Gerenciar a qualidade dos dados exibidos, para garantir decisões precisas
- Integrar dados de diferentes fontes, inclusive dados em tempo real
- Traçar uma linha de raciocínio clara na análise dos dados, com técnicas de storytelling
- Integrar os painéis gerenciais nas atividades diárias com plataformas móveis e dashboards

O propósito da visualização de dados não é criar gráficos bonitos, mas sim ser usada como ferramenta estratégica para tomada de decisão.

Escolhendo o Melhor Design para a Visualização de Dados

As imagens são processadas mais rapidamente pelo cérebro.

A imagem ajuda na compreensão de dados complexos. Isso porque ajudam a identificar padrões e contam histórias.

A escolha do melhor design para as visualizações de dados, vai muito além de questões técnicas. O fator humano, o público e os objetivos são determinantes na seleção da visualização que será adotada. Mas a sua capacidade de contar a história dos dados será muito mais relevante que qualquer gráfico que você utilize.

Design Thinking

É uma abordagem formada no campo do design e adaptada a empresas e corporações. Literalmente, o termo significa pensamento em design.

O design tem como objetivo a promoção do bem-estar na vida das pessoas.

Da mesma forma que um profissional do design enxerga de forma holística o mundo ao seu redor, observando aspectos cognitivos, emocionais e estéticos que afetam as experiências humanas, empresários precisam olhar seu contexto com empatia, a fim de identificar problemas a serem solucionados, bem como criar respostas verdadeiramente inovadoras para eles.

O foco passa a ser a experiência do consumidor ou do público-alvo, na busca por respostas aos problemas encontrados através do Design Thinking enquanto abordagem metodológica.

Fases do Design Thinking

1. Descoberta – Eu tenho um desafio! Como posso abordá-lo?
2. Interpretação – Eu aprendi alguma coisa! Como posso interpretá-la?
3. Ideação – Eu tenho uma oportunidade! Como posso criá-la?
4. Experimentação – Eu tenho uma ideia! Como posso concretizá-la?
5. Evolução – Eu experimentei alguma coisa! Como posso aprimorá-la?
6. Repetição – Eu aprimorei alguma coisa! Como posso repeti-la?

Cientista de Dados – O Contador de Histórias

Com a massiva quantidade de dados aumentando a cada dia, um grande desafio vem surgindo para aqueles responsáveis por analisar, sumarizar e apresentar os dados: fazer com que a informação gerada, possa ser facilmente compreendida. E uma das tarefas mais importantes do trabalho do Cientista de Dados, é ser capaz de transmitir tudo aquilo que os dados querem dizer. E às vezes os dados querem dizer coisas diferentes, para públicos diferentes. Pode parecer fácil, a princípio. Hoje temos à nossa disposição os mais variados recursos para apresentação e exatamente aí que está o desafio. Nunca foi tão fácil gerar tabelas e gráficos, com diferentes estruturas, formatos, tamanhos, cores e fontes. Os gráficos estão deixando de ser gráficos e se tornando infográficos. Ter um volume cada vez maior de dados à nossa disposição, não torna mais fácil a apresentação da informação gerada. Pelo contrário, torna a tarefa mais complicada. Quase uma arte.

Desde cedo e ao longo da vida acadêmica, aprendemos sobre números e texto. Na prática, o ser humano pratica análise de dados, desde os primeiros anos de vida. Mas uma habilidade cada vez mais fundamental no universo de Data Science (e me arrisco dizer, em qualquer profissão) é a habilidade de contar histórias a partir de dados. A tecnologia está permitindo analisar cada vez mais dados e cada vez mais rápido. E ser capaz de visualizar os dados e contar a história que eles querem dizer, é uma habilidade cada vez mais valorizada. Visualizar os dados e contar a história, é a chave que pode levar a decisões cada vez mais precisas.

Tenho visto profissionais de ciência de dados confiarem apenas na tecnologia para explicar o que os dados querem dizer. Hoje, qualquer um pode colocar alguns dados em uma planilha Excel, criar um gráfico e apresentar uma informação. Para muitos, o processo de visualização de dados termina aqui. Diversas ferramentas do mercado criam dashboards magníficos, com alguns poucos cliques. Mas os dados possuem uma história e as ferramentas tecnológicas não sabem como contar a história, pelo menos não ainda. Exatamente aí que o Cientista de Dados precisa ser um contador de histórias. Precisa ser mais que um analista, precisa ser um comunicador, capaz de dar vida a história e contextualiza-la no mundo real. Sem a habilidade de contar a história dos dados, estaremos apenas mostrando os dados.

Uma visualização efetiva de dados, pode ser a diferença entre sucesso e falha nas decisões de negócio. Em breve, a capacidade de comunicar e contar as histórias dos dados, será uma das características mais valorizadas e buscadas pelas empresas. Os profissionais de dados possuem habilidades com números, programação e estatística, mas poucos possuem habilidades efetivas de comunicação.

É incrível poder usar a ciência de dados, na tomada de decisões. Mas ao longo do tempo percebemos, que comunicar corretamente o que os dados querem dizer, é um fator de sucesso que faz a diferença. Por isso, técnicas de design começam a ser incorporadas ao mundo de Data Science, ou seja, pensar o design do produto final (a história), de forma que a comunicação seja a mais efetiva possível. Ao pensar na história dos dados que queremos contar como um produto, temos condição de pensar o processo de forma reversa, ou seja, pensamos primeiro no usuário final, no consumidor daquela informação. Como o usuário final vai utilizar a informação contida nos dados? Qual a melhor forma de mostrar isso a ele? Qual a formação, experiência e conhecimento prévio daquele grupo de usuários? São tomadores de decisão ou formadores de opinião? É um grupo homogêneo ou heterogêneo? O processo de comunicação dos dados é realizado levando em consideração como o usuário que ouvirá a história, irá interpretá-la. Isso não significa falar com o

que a pessoa quer ouvir, mas sim mostrar o que os dados querem dizer, de forma que a compreensão seja clara e conectada com a realidade do público alvo. Para as pessoas que não têm muita paciência com analytics, contar uma história pode ter grande influência na forma como se apresenta a informação certa no formato certo. Uma declaração como “52.000 pessoas foram assassinadas no Brasil em 2014 e em 2015 esse número deve aumentar em 2.5%” pode captar muito mais a atenção de alguém ao invés de apresentar o mesmo comunicado através de gráficos que mostrem as tendências de crimes ao longo dos anos, devido a problemas socioeconômicos.

Um contador de histórias de dados é um perito que analisa os dados disponíveis na forma de gráficos, diagramas ou qualquer outro tipo de representação visual, processa as informações a partir dos dados analisados para compreender o que isso significa para um determinado setor, organização ou marca e, em seguida, fornece insights sobre a forma de uma história. Um contador de histórias de dados descobre o ponto de vista mais convincente para orientar seus usuários a tomar uma decisão lógica, que é suportada com provas e não apenas com opinião de especialistas. Contar histórias com dados não se limita a compreensão de vários conceitos estatísticos avançados, mas também requer a compreensão sobre o comportamento humano.

O Cientista de Dados deve ser um contador de histórias e deve ser capaz de contar a mesma história de maneiras diferentes. O profissional que for capaz de unir as habilidades técnicas necessária para análise de dados, com a capacidade de contar histórias, será o verdadeiro unicórnio, aquele profissional único.

9.2. Métodos de Visualização

Por Que o Cérebro Precisa de Visualização de Dados?

O cérebro humano é incapaz de processar mais de um valor a cada vez, imagine centenas, milhares, milhões ou bilhões de valores.

O objetivo da visualização é simplificar o valor dos dados, promover a compreensão sobre eles, e comunicar conceitos e ideias importantes.

A visualização de dados avançadas suportam técnicas analíticas mais profundas e complexas.

A visualização de dados apresenta os dados de uma forma que um diretor pode facilmente interpretar, poupando-lhe tempo e energia.

Independentemente da sua experiência com computadores um diretor pode, rapidamente, extrair conhecimento a partir dos dados, utilizando gráficos.

A longo prazo, esta colaboração pode poupar tempo e ajudar a combater algumas das falhas do processo decisório entre áreas de negócio.

Quais Gráficos Escolher para Sumarizar Meus Dados?

É fundamental que sua visualização tenha uma finalidade e que você seja seletivo quanto ao que incluir nesta visualização para atender a um objetivo.

A primeira pergunta a ser feita para escolher um bom gráfico é:

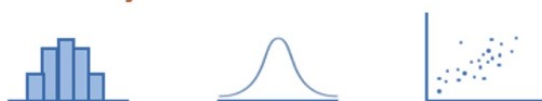
O que será mostrado no gráfico?

- Comparação: confrontar dados ao longo do tempo ou entre várias categorias
- Distribuição: mostrar a frequência em que ocorrem os dados
- Relação: mostrar a interdependência entre variáveis
- Composição: mostrar os componentes de um todo

Comparação:



Distribuição:



Relação:



Composição:



A segunda pergunta a ser feita para escolher um bom gráfico é:

Quantas variáveis, itens ou categorias serão mostrados no gráfico?

- Comparação:
 - Confrontar ao longo do tempo
 - Muitos períodos:

- Cíclicos: radar
- Não Cíclicos (lineares): Linha
- Poucos períodos:
 - Poucas categorias: colunas
 - Muitas categorias: linhas
- Entre itens
 - Poucos itens: colunas
 - Muitos itens: barras
- Composição:
 - Mostrar componentes de um tudo
 - Varia ao longo do tempo:
 - Poucos períodos:
 - Apenas diferenças relativas importam: colunas empilhadas (100%)
 - Diferenças relativas e absolutas importam: colunas
 - Muitos períodos:
 - Apenas diferenças relativas importam: áreas empilhadas (100%)
 - Diferenças relativas e absolutas importam: áreas empilhadas
 - Estática:
 - Porção simples do total: pizza
 - Acumulação e subtração do total: cachoeira
- Distribuição:
 - Mostrar a frequência em que ocorrem os dados:
 - Variável única:
 - Poucos dados: histograma de colunas
 - Muitos dados: histograma de linha
 - Duas variáveis: dispersão
- Relação:
 - Mostrar a interdependência entre variáveis:
 - Duas variáveis: dispersão
 - Três variáveis: bolhas

A terceira pergunta a ser feita para escolher um bom gráfico é:

Quem vai interpretar e analisar o gráfico?

- Quem precisa ler o gráfico?

Tudo Depende dos Tipos de Dados?

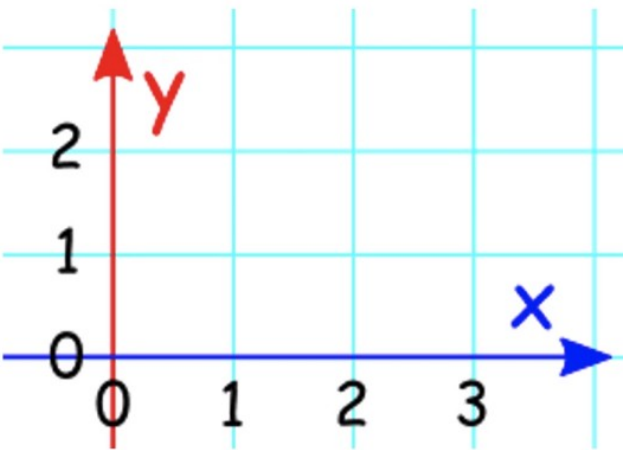


















Tipos de Dados (Tipos de Variáveis):

- Qualitativas:
 - Nominais: profissão, sexo, religião
 - Ordinais: escolaridade, classe social, fila
- Quantitativas (numéricas):
 - Discretas: número de filhos, de carros, de acessos
 - Contínuas: altura, peso, salário

O Que São e Como Usar Visual Encodings?

Visual encoding ou codificação visual é a maneira pela qual os dados são mapeados em estruturas visuais, sobre as quais construímos as imagens em uma tela.

Existem 2 categorias de visual encodings:

Planar: x e y	Retinal: tamanho (size), orientação (orientation), saturação de cor (color saturation), matiz de cores (color hue), formato (format) e textura (texture)						
	<table><tr><td>Size </td><td>Color Hue </td></tr><tr><td>Orientation </td><td>Shape </td></tr><tr><td>Color Saturation </td><td>Texture </td></tr></table>	Size 	Color Hue 	Orientation 	Shape 	Color Saturation 	Texture 
Size 	Color Hue 						
Orientation 	Shape 						
Color Saturation 	Texture 						

Processo de Criação de Charts e Plots

Chart é um gráfico mais elaborado. Já o Plot é um gráfico bem mais simples.

Um gráfico possui 3 camadas (Data Visualization Framework): dados, mapeamento e gráfica.

Data Layer	Mapping Layer	Graphics Layer
Localização e obtenção de dados; Importação de dados no formato apropriado; Relacionamento entre os dados	Análise de dados e algoritmos; Associação de forma apropriada dos diferentes canais de dados; Geometria e correspondência.	Gerenciamento das interações; Conversão da geometria dos dados em imagens; Decoração; Visual encodings.

para correspondência apropriada; Análise de dados e agregação.		
-------------------------------------------------------------------	--	--

O Que é Área de Plotagem?

Área de plotagem é a área onde os gráficos são mostrados.

Técnicas de Apresentação

Poucos medos são tão comuns e populares quanto o de falar em público. Seja na universidade, na escola, apresentando projetos na empresa, workshops e qualquer outra apresentação em geral, o medo de ter que encarar o público é sempre um dos fatores que mais atrapalham. Não é à toa que existem tantas dicas que ajudam nesses momentos. Fazer apresentações é algo que está presente na rotina de um Cientista de Dados. Apresentar em público parece uma tarefa difícil, mas não é. Pensando nisso, iremos listar algumas técnicas de apresentação para ajudá-lo nesse processo. Confira!

Não tenha dúvida, não existe técnica de apresentação melhor do que estudar muito o tema do qual falará. O segredo é a confiança: quanto mais você estiver confortável com o assunto, mais confiança você passa para sua plateia. Mas, para conquistar isso, talvez, a resposta não seja apenas aprender sobre o tema, mas sim saber o que fazer com esse conhecimento. Uma das dicas principais é sempre se preocupar com que sua plateia saia da apresentação sabendo de algo que ela não sabia. Não adianta falar aquilo que todo mundo sabe com um toque diferenciado. Não adianta não desafiar seus espectadores. Não deixe que tudo se torne um sonífero. Questione, faça perguntas, observe as respostas e use-as para acrescentar novos rumos e exemplos em sua palestra ou apresentação.

E isso só é possível quando você tem o domínio do assunto. E organizar esses pensamentos é o caminho mais curto até o sucesso. Você pode, por exemplo, criar um “mapa mental” para isso. Seja digital, em papel ou só dentro de sua cabeça, o importante é saber as opções de caminhos a cada curva. Para dar ainda mais naturalidade para demonstrar este conhecimento, não decore ordens e esquemas mecânicos. Siga seu caminho com segurança. Mas nem por isso deixe de lado as ferramentas que te ajudam nesse caminho. Lembre-se sempre que você é um especialista na sua área. Você tem conhecimento para falar sobre isso. Com um pouco de preparo, essa, que é uma das principais técnicas de apresentação, será facilmente realizada!

Para deixar sua apresentação mais impactante, você deve usar recursos multimídias como o Power Point, e até recursos visuais mais simples se necessário. O ouvinte guarda cerca de 10% do que foi falado na palestra três dias depois (comprovado cientificamente). Porém, quando utilizado conteúdo multimídia, esse número salta para 65%. O segredo para aumentar o impacto deste conteúdo é pensar que aquele conteúdo multimídia é apenas um guia, e não um texto ou um livro. Isso tudo te ajuda a impactar mais seu público. Não adianta escrever tudo em slides e passar uma hora de costas para a plateia lendo uma projeção na parede. É preciso ter em mente que a atração principal, as maiores informações e todas descobertas devem vir de você, e não de um computador.

Organize seus slides, ou qualquer outro recurso, de modo limpo e claro. Nada de textos enormes, foque tudo em tópicos de fácil entendimento e que ajudem sua plateia a percorrer o conteúdo com você. Por outro lado, use esses tópicos para seguir sua linha de pensamento, mas nunca esqueça de

incrementar essas informações. Não exagere no número de slides e muito menos na quantidade de informações em cada um deles.

Procure um balanço entre esses dois fatores, mas sempre que possível use imagens. Prefira boas imagens no lugar de qualquer texto. Uma imagem prende muito mais a atenção do que meia dúzia de tópicos, o que torna toda a apresentação muito mais ágil e marcante.

Vejam agora algumas outras dicas separadas em 2 tópicos principais: Preparação e Contato com o Público.

Preparação

- Procure estudar o tema com antecedência! O domínio ajuda você ter segurança na hora da apresentação.
- Prepare um esquema com o seu raciocínio sobre a apresentação. Cada slide deve ser um tópico, escreva as palavras e termos que estão associados.
- Procure criar slides simples e use imagens que prendam a atenção do ouvinte. Poucas palavras somente para dar o link.
- Procure caracterizar a apresentação com início, meio e fim. Quando for apresentar explore a estrutura e apresente aos participantes.

Contato com o Público

- Fique atento à gramática e vocabulário. Um bom vocabulário evita termos pobres e vulgares, como palavrões e gírias. Mas também evite vocabulários complicados muitas vezes incompreensíveis.
- Preocupe-se em pronunciar bem e completamente as palavras, olhe nos olhos dos ouvintes e procure dar atenção para todos.
- Fale com ritmo, de acordo com seu objetivo. Se for uma aula deve ser mais pausada, se for uma palestra motivacional deve ter um tom mais emocionante, desta forma conseguirá uma comunicação agradável e transmitirá congruência ao discurso.
- É fundamental que você tenha o seu próprio estilo, que se sinta confortável e confiante consigo próprio de forma a transmitir segurança aos seus ouvintes.

O que você deve evitar e o que você deve fazer em uma apresentação:

Evite:

- Mãos nos bolsos ou atrás das costas e cruzar os braços.
- Procure ficar em pé, nunca debruce sobre a mesa, cadeira ou tribuna.
- Colocar-se à frente do projetor.
- Movimentar-se desordenadamente.
- Relaxar a sua postura de tronco e ombros caídos.

Faça:

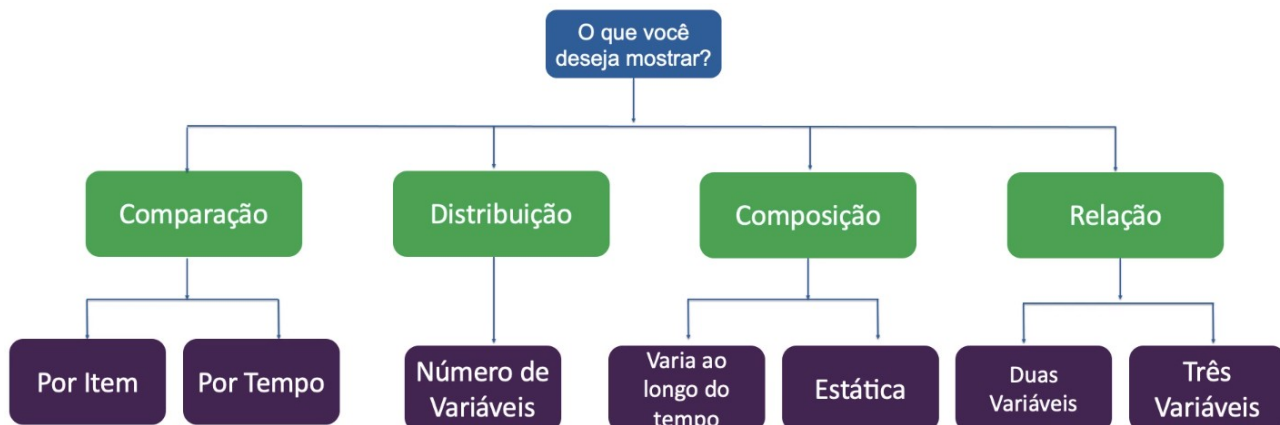
- Sorria.
- Gesticule com moderação.
- Estabeleça contato visual com a plateia.
- Procure tratar os participantes pelo nome e envolva os participantes.
- Lembre-se sempre que a apresentação tem início, meio e fim.

Principais Métodos de Visualização

São 4 categorias principais de métodos de visualização:





- Infográficos
- Mapas Mentais e Organogramas
- Gráficos e Tabelas
- Modelos Matemáticos

Visualização de Dados Numéricos ou Não Numéricos



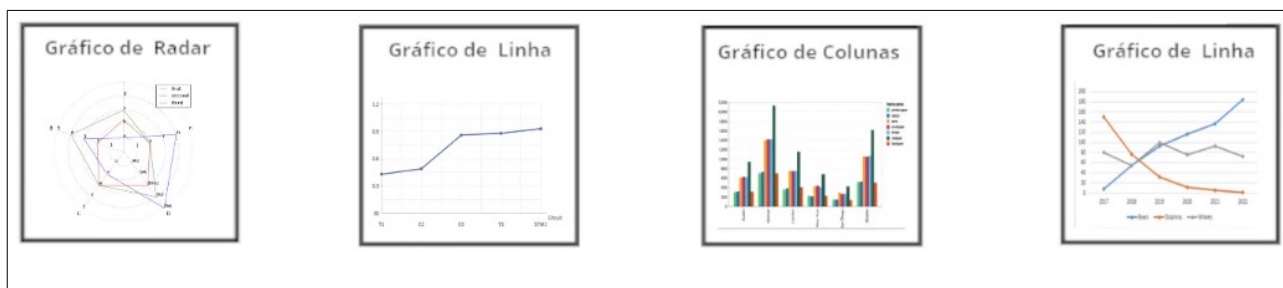
Comparação:

O que você deseja mostrar: **Comparação por item**

Gráfico de Colunas de Largura Variável	Tabela com Gráficos Incorporados	Gráfico de Barras	Gráfico de Colunas
Duas variáveis por item	Uma variável por item, várias categorias	Uma variável por item, vários itens, poucas categorias	Uma variável por item, poucos itens, poucas categorias
			

O que você deseja mostrar: **Comparação por tempo**

Gráfico de Radar	Gráfico de Linha	Gráficos de Colunas	Gráfico de Linhas
Várias observações, dados cíclicos	Várias observações, dados não cíclicos	Poucas observações, poucas categorias	Poucas observações, várias categorias



Distribuição:

O que você deseja mostrar: **Número de variáveis**

Histograma de coluna	Histograma de linha	Gráfico de dispersão	Gráfico de superfície
Variável única, poucas observações	Variável única, muitas observações	Duas variáveis	Três variáveis

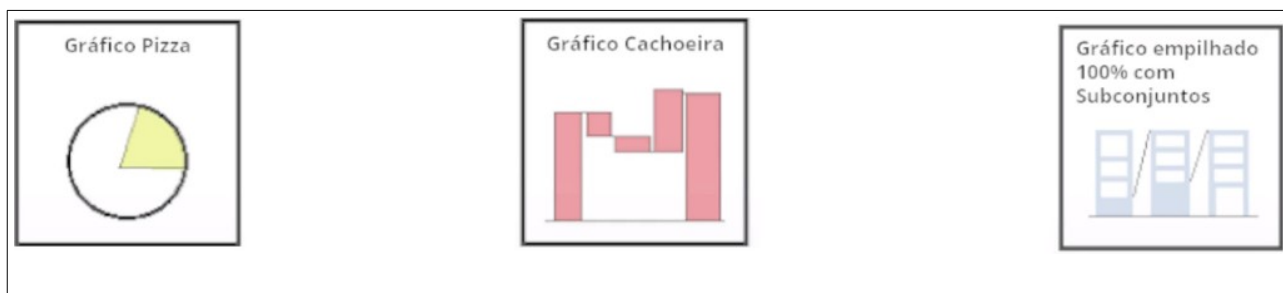
Composição:

O que você deseja mostrar: **Varia ao longo do tempo**

Gráfico de Colunas Empilhadas 100%	Gráfico de Colunas Empilhadas	Gráfico de Área Empilhado 100%	Gráfico de Área Empilhado
Poucas observações, apenas diferenças relativas	Poucas observações, diferenças relativas e absolutas importam	Muitas observações, apenas diferenças relativas importam	Muitas observações, diferenças relativas e absolutas importam



O que você deseja mostrar: **Estática**

Gráfico Pizza	Gráfico Cachoeira	Gráfico empilhado 100% com subconjuntos
Simples fração do total	Acumulação ou subtração do total	Conjuntos e subconjuntos



Relação:

O que você deseja mostrar: ***Duas ou Três Variáveis***

Gráfico de Dispersão	Gráfico de Bolhas
Duas variáveis	Três variáveis
	

Ferramentas de Visualização de Dados

A visualização de dados é baseada em vários dos mesmos princípios do web design. Web design deve ser útil e agradável. Um site agradável nos atrai naturalmente, mas é quando extraímos algum valor dele que decidimos ficar ali (e não apenas passar por ele).

As mesmas regras se aplicam à visualização de dados: é sobre apresentar todos os nossos dados de modo que sejam fáceis de entender e intuitivos para navegar – interação é fundamental. Nós reunimos mais e mais informações todos os dias, e isso levou à incrível popularidade dos infográficos – um jeito fácil para “digerirmos” números que, sem uma representação gráfica, simplesmente parecem chatos! Já que nossos cérebros são capazes de processar imagens a uma velocidade maior do que processam números, todo conteúdo em sites está se tornando mais e mais visualmente “data-driven” e tabelas, gráficos de pizza e gráficos de barra são agora elementos comuns para todo web designer. Se você quer criar visualizações de dados incríveis, você precisará das melhores ferramentas.

Listamos aqui as ferramentas que consideramos mais relevantes no mercado atual e muitas delas usaremos ao longo do curso.

- Tableau
- Power BI
- Qlik Sense
- SAS Visual Analytics
- Plotly
- D3.js
- Visually

- Chart.js
- Leaflet
- Polymaps
- Linguagem R e seus diversos pacotes gráficos
- Linguagem Python e seus diversos pacotes gráficos

Para as aulas práticas deste capítulo usaremos Linguagem Python com o pacote gráfico Plotly, que nos permite criar gráficos atraentes, de qualidade e de forma simples.

Glossário de Formatação dos Gráficos

Você vai perceber que os gráficos são quase todos criados com o mesmo padrão em Linguagem Python e com Plotly. Aqui está um glossário para ajudar você a identificar mais rapidamente a formatação dos gráficos:

Plotly express: funções que podem criar figuras inteiras de uma vez. É o ponto de partida recomendado para a criação de figuras mais comuns.

Graph Objects: as figuras criadas, manipuladas e renderizadas pela biblioteca Python plotly são representadas por estruturas de dados em forma de árvore que são serializadas automaticamente em formato JSON para renderização pela biblioteca JavaScript Plotly.js.

Elementos básicos no layout:

- `xaxis_title`: rótulo do eixo x
- `yaxis_title`: rótulo do eixo y
- `title`: título do plot
- `title_font_size`: tamanho da fonte do título do gráfico
- `height`: altura do gráfico
- `width`: largura do gráfico
- `show_legend`: habilitar legenda
- `xaxis_type` / `yaxis_type`: Tipo de eixo X / Y
- `xaxis_showgrid` / `yaxis_showgrid`: Exibir grades ou não
- `gridcolor`: cor da grade • `gridwidth`: largura da grade

Elementos básicos em eixos:

- `showticklabels`: exibe rótulos de escala ou não
- `tickangle`: ângulo dos rótulos de escala
- `tickfont`: fonte dos rótulos de escala
- `tickprefix`: prefixo das etiquetas do tick
- `showline`: linha de contorno do gráfico
- `linewidth`: largura da linha do contorno do gráfico
- `linecolor`: cor da linha do contorno do gráfico
- `mirror`: lado oposto da área de plotagem
- `intervalo`: limite de intervalo do eixo

Gráficos Padrões com Plotly

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Visualização de Dados e Design de Dashboards</font>
## <font color='blue'>Capítulo 2 - Métodos de Visualização</font>
```

```

## Exemplos de Gráficos Padrões
Aqui você encontra exemplos de alguns dos gráficos padrões estudados no Capítulo 2.
![title](imagens/cap02.png)
## Pacotes Python Para Manipulação de Dados e Visualização
Certifique-se de usar os pacotes nas mesmas versões mostradas abaixo.
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark

# ->
# Manipulação de Dados
import numpy as np
import pandas as pd
# Visualização com Plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.colors import n_colors
from plotly.subplots import make_subplots

# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions

## Carregando os Dados
Consulte as fontes dos dados no manual em pdf **Gráficos Padrões com Plotly** no Capítulo 2.
# ->
titanic = pd.read_csv('dados/titanic-train.csv')
titanic.head(3)

# ->
netflix = pd.read_csv("dados/netflix-shows-netflix_titles.csv")
netflix.head(3)

# ->
world = pd.read_csv('dados/world-university-rankings-cwurData.csv')
world.head(3)

# ->
google = pd.read_csv("dados/google-play-store-apps-googleplaystore.csv")
google.head(3)

# ->
covid = pd.read_csv('dados/novel-corona-virus-2019-dataset-covid_19_data.csv')
covid.head(3)

# ->
covid_line = pd.read_csv('dados/novel-corona-virus-2019-dataset-COVID19_line_list_data.csv')
covid_line.head(3)

### Gráfico de Barras Simples
**Objetivo**: Exibe a representação quantitativa de uma variável.
Quais países tem universidades com pontuação superior a 70?
# ->
# Prepara os dados
top_countries = world[world['score'] > 70]['country'].value_counts().reset_index().rename(columns = {'index':'country','country':'count'})

# ->
# Cria a figura
fig = go.Figure(go.Bar(x = top_countries['country'], y = top_countries['count'],))
# Configura o layout
fig.update_layout(title_text = 'Países com Universidades com Pontuação Superior a 70',
                    xaxis_title = "País",
                    yaxis_title = "Número de Universidades")
# Mostra o gráfico
fig.show()

### Gráfico de Barras - Gradiente de Cores e Posicionamento de Texto
**Objetivo**: Exibe a representação quantitativa de uma variável destacando as contagens com gradiente de cor e posição do texto para todas as barras.
Em que gênero se enquadra a maioria dos aplicativos da Google Play Store?

```

```

# ->
# Prepara os dados
apps = google['Genres'].value_counts()[1:10].to_frame().reset_index().rename(columns = {'index':'Genres','Genres':'Count'})

# ->
# Cria a figura e formata o gradiente de cores
fig = go.Figure(go.Bar(x = apps['Genres'],
                        y = apps['Count'],
                        marker = {'color': apps['Count'], 'colorscale': 'Viridis'},
                        text = apps['Count'],
                        textposition = "outside",))
# Configura o layout
fig.update_layout(title_text = 'Top Apps da Google Playstore',
                  xaxis_title = "Gêneros de Aplicativos",
                  yaxis_title = "Número de Apps")
# Mostra o gráfico
fig.show()

### Gráfico de Barras Empilhado
**Objetivo**: Exibe a representação quantitativa de uma variável agrupando / empilhando as barras.
Quanto programas / filmes foram lançados na Netflix do Brasil e dos Estados Unidos entre 2015 e 2020?
# ->
# Agrupamento dos dados
top_release_brazil = netflix[(netflix['country']=='Brazil')&
                             ((netflix['release_year']==2015)|(netflix['release_year']==2016)|(netflix['release_year']==2017)|(netflix['release_year']==2018)|
                              (netflix['release_year']==2019)|(netflix['release_year']==2020))]
[release_year'].value_counts().to_frame().reset_index().rename(columns={'index':'release_year','release_year':'count'})
top_release_us = netflix[(netflix['country']=='United States')&
                         ((netflix['release_year']==2015)|(netflix['release_year']==2016)|(netflix['release_year']==2017)|(netflix['release_year']==2018)|
                          (netflix['release_year']==2019)|(netflix['release_year']==2020))]
[release_year'].value_counts().to_frame().reset_index().rename(columns={'index':'release_year','release_year':'count'})

# ->
# Cria a figura
fig = go.Figure()
# Formata as barras por agrupamento
fig.add_trace(go.Bar(x = top_release_brazil[release_year],
                     y = top_release_brazil[count],
                     name = 'Brasil',
                     marker_color = 'violet'))
fig.add_trace(go.Bar(x = top_release_us[release_year],
                     y = top_release_us[count],
                     name = 'Estados Unidos',
                     marker_color = 'blue'))
# Formata o layout
fig.update_layout(title_text = 'Filmes e Programas Lançados na Netflix do Brasil / EUA Entre 2015 e 2020',
                  xaxis_title = "Ano",
                  yaxis_title = "Número de Filmes/Programas",
                  barmode = 'stack')
# Mostra o gráfico
fig.show()

### Gráfico de Barras Facetado
**Objetivo**: Exibe uma visão de diferentes características categóricas em relação a uma única variável numérica.
Qual é a soma da tarifa (variável fare) por sexo dos passageiros em cada classe e seus embarcados no dataset Titanic?
- Variáveis facetárias: Survived, Pclass
- Barras agrupadas: Embarked
- Numérico único (eixo Y) - Fare
# ->
# Prepara os dados
facet_titanic = titanic[['Sex','Survived','Embarked','Pclass','Fare']].groupby(['Sex','Survived','Embarked','Pclass']).agg('sum').reset_index()

# ->
# Cria o facet
fig = px.bar(facet_titanic,
             x = "Sex",
             y = "Fare",
             color = "Embarked",
             barmode = "group",
             facet_row = "Survived",
             facet_col = "Pclass",)
# Layout
fig.update_layout(title_text = 'Vista Facetada da tarifa dos passageiros do Titanic em relação à idade, classe, embarque\n')
# Mostra o gráfico
fig.show()

### Gráfico de Barras Horizontal
**Objetivo**: Exibe a representação quantitativa de uma variável de maneira horizontal.

```


Quantos aplicativos da Playstore se enquadram em cada categoria?

```
# ->
# Prepara os dados
app_category = google['Category'].value_counts()[1:15].reset_index().rename(columns = {'index':'Category', 'Category':'Count'}).sort_values('Count',
ascending = "False")
```

```
# ->
# Cria a figura com orientação horizontal
fig = go.Figure(go.Bar(y = app_category['Category'], x = app_category['Count'], orientation = "h"))
# Layout
fig.update_layout(title_text = '15 Principais Categorias de Aplicativos da Google Playstore',
                    xaxis_title = "Total de Apps",
                    yaxis_title = "Categoria")
# Mostra o gráfico
fig.show()
```

Gráfico de Linhas

****Objetivo**:** Relação entre variáveis ao longo do tempo.

Quantas mortes por COVID foram observadas ao longo do tempo?

```
# ->
# Prepara os dados
total_confirmed = covid[['ObservationDate', 'Deaths']].groupby('ObservationDate').sum().reset_index()
```

```
# ->
# Cria a figura
fig = go.Figure(data = go.Scatter(x = total_confirmed['ObservationDate'],
                                  y = total_confirmed['Deaths'],
                                  mode = 'lines'))
# Layout
fig.update_layout(title = 'Número de Mortes Por COVID ao Longo do Tempo',
                    xaxis_title = "Data",
                    yaxis_title = "Número de Casos")
# Mostra o gráfico
fig.show()
```

Gráfico de Pizza

****Objetivo**:** Exibe representação quantitativa (contagem) e proporcional de variáveis categóricas.

Qual a proporção de tipos de programas da Netflix?

```
# ->
# Prepara os dados
net_category = netflix['type'].value_counts().to_frame().reset_index().rename(columns = {'index':'type', 'type':'count'})
```

```
# ->
# Cria a figura
fig = go.Figure([go.Pie(labels = net_category['type'], values = net_category['count'])])
# Interatividade
fig.update_traces(hoverinfo = 'label+percent',
                  textinfo = 'value+percent',
                  textfont_size = 15,
                  insidetextorientation = 'radial')
# Layout
fig.update_layout(title = "Tipos de Programas na Netflix", title_x = 0.5)
# Gráfico
fig.show()
```

Gráfico de Pizza com Cores Customizadas

****Objetivo**:** Exibe representação quantitativa em pizza com cores personalizadas para os rótulos.

Qual a proporção de passageiros do Titanic por faixa etária?

```
# ->
# Prepara os dados
titanic = titanic.dropna()
titanic['age_category'] = np.where((titanic['Age'] < 19), "Abaixo de 19 Anos",
                                  np.where((titanic['Age'] > 18)&(titanic['Age'] <= 30), "19-30 Anos",
                                  np.where((titanic['Age'] > 30)&(titanic['Age'] <= 50), "31-50 Anos",
                                  np.where(titanic['Age'] > 50, "Acima de 50 Anos", "NULL"))))
age = titanic['age_category'].value_counts().to_frame().reset_index().rename(columns = {'index':'age_category', 'age_category':'Count'})
titanic_age = titanic['age_category'].value_counts().to_frame().reset_index().rename(columns = {'index':'age_category', 'age_category':'count'})
```

```
# ->
# Lista de cores
colors = ['green', 'violet', 'yellow', 'blue']
```

```
# ->
# Figura
fig = go.Figure([go.Pie(labels = titanic_age['age_category'], values = titanic_age['count'])])
# Interatividade
fig.update_traces(hoverinfo = 'label+percent',
                  textinfo = 'percent+label',
```

```

        textfont_size = 16,
        marker = dict(colors = colors, line = dict(color = '#000360', width = 2)))
fig.update_layout(title = "Passageiros do Titanic Por Faixa Etária", title_x = 0.5)
fig.show()

#### Gráfico de Rosca (Donut)
**Objetivo**: Exibe representação quantitativa em formato de rosca.
Qual é a contagem de distribuição da classificação de conteúdo de aplicativos do google playstore?
# ->
# Prepara os dados
content = google['Content Rating'].value_counts().to_frame().reset_index().rename(columns = {'index':'Content Rating','Content Rating':'count'})

# ->
# Figura
fig = go.Figure([go.Pie(labels = content['Content Rating'], values = content['count'], hole = 0.3)])
# Interatividade
fig.update_traces(hoverinfo = 'label+percent',
                  textinfo = 'percent',
                  textfont_size = 12)
# Layout
fig.update_layout(title = "Classificação de Conteúdo do Google Apps", title_x = 0.5)
# Gráfico
fig.show()

# Fim

```

Gráficos Comparativos com Plotly

```

# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Visualização de Dados e Design de Dashboards</font>
## <font color='blue'>Capítulo 2 - Métodos de Visualização</font>
## Exemplos de Gráficos Comparativos
Aqui você encontra exemplos de alguns dos gráficos comparativos estudados no Capítulo 2.
![title](imagens/cap02.png)
## Pacotes Python Para Manipulação de Dados e Visualização
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark

# ->
# Manipulação de Dados
import numpy as np
import pandas as pd
# Plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.colors import n_colors
from plotly.subplots import make_subplots

# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions

## Carregando os Dados
# ->
# Carrega os dados
titanic = pd.read_csv('dados/titanic-train.csv')

#### Bubble Plot
**Objetivo**: Exibe representação quantitativa destacando a categoria mais ocorrida com o tamanho da bolha.
Quantas pessoas viajaram em cada classe de passageiros do Titanic?
# ->
# Prepara os dados
pclass = titanic['Pclass'].value_counts().to_frame().reset_index().rename(columns = {'index':'Pclass','Pclass':'Count'})

# ->

```

```

# Figura
fig = go.Figure(data = [go.Scatter(x = pclass['Pclass'],
                                y = pclass['Count'],
                                mode = 'markers',
                                marker = dict(size = pclass['Count']*0.3))])

# Layout
fig.update_layout(title = 'Viajantes Por Classe do Titanic',
                  xaxis_title = "Classe",
                  yaxis_title = "Número de Pessoas")

# Mostra o gráfico
fig.show()

### Bubble Plot Com Gradiente de Cor
**Objetivo** : Exibe uma representação quantitativa destacando a categoria mais ocorrida com o gradiente de cor da bolha.
Quantas pessoas viajaram no Titanic por faixa etária?
# ->
# Prepara os dados
titanic = titanic.dropna()
titanic['age_category'] = np.where((titanic['Age'] < 19), "Abaixo de 19 Anos",
                                np.where((titanic['Age'] > 18) & (titanic['Age'] <= 30), "19-30 Anos",
                                np.where((titanic['Age'] > 30) & (titanic['Age'] <= 50), "31-50 Anos",
                                np.where(titanic['Age'] > 50, "Acima de 50 Anos", "NULL"))))
age = titanic['age_category'].value_counts().to_frame().reset_index().rename(columns = {'index':'age_category','age_category':'Count'})

# ->
# Figura
fig = go.Figure(data = [go.Scatter(x = age['age_category'],
                                y = age['Count'],
                                mode = 'markers',
                                marker = dict(color = age['Count'],
                                size = age['Count'],
                                showscale = True))])

# Layout
fig.update_layout(title = 'Viajantes do Titanic Por Faixa Etária',
                  xaxis_title = "Faixa Etária",
                  yaxis_title = "Número de Pessoas")

# Gráfico
fig.show()

## Waterfall
**Objetivo** : Exibir aumento e diminuição de contagens no gráfico em cascata
Apresente um resumo dos casos de Covid nos meses de junho e julho nos Estados Unidos.
# ->
# Carrega os dados
covid = pd.read_csv('dados/novel-corona-virus-2019-dataset-covid_19_data.csv')

# ->
# Ajusta coluna de data
covid['ObservationDate'] = pd.to_datetime(covid['ObservationDate'])
covid['month'] = covid['ObservationDate'].dt.month_name()

# ->
# Prepara os dados
us_confirmed_july = covid[(covid['Country/Region']=='US') & (covid['month']=='July')]
[['month','Confirmed']].groupby('month').sum().reset_index()['Confirmed']
us_confirmed_june = covid[(covid['Country/Region']=='US') & (covid['month']=='June')]
[['month','Confirmed']].groupby('month').sum().reset_index()['Confirmed']
us_recovered_july = covid[(covid['Country/Region']=='US') & (covid['month']=='July')]
[['month','Recovered']].groupby('month').sum().reset_index()['Recovered']
us_recovered_june = covid[(covid['Country/Region']=='US') & (covid['month']=='June')]
[['month','Recovered']].groupby('month').sum().reset_index()['Recovered']
us_death_july = covid[(covid['Country/Region']=='US') & (covid['month']=='July')][['month','Deaths']].groupby('month').sum().reset_index()
['Deaths']
us_death_june = covid[(covid['Country/Region']=='US') & (covid['month']=='June')][['month','Deaths']].groupby('month').sum().reset_index()
['Deaths']

# ->
# Figura
fig = go.Figure(go.Waterfall(name = "Casos de Covid",
                            orientation = "v",
                            measure = ["relative",
                            "relative",
                            "total",
                            "relative",
                            "relative",
                            "total"],
                            x = ["June Confirmed",
                            "July Confirmed",

```

```

        "Total Confirmed",
        "June Recoverd & Deaths",
        "July Recovered & Deaths",
        "Total Under Observation"],
y = [int(us_confirmed_june),
      int(us_confirmed_july),
      0,
      -(int(us_death_june) + int(us_recovered_june)),
      -int((us_death_july) + int(us_recovered_july)),
      0],
connector = {"line": {"color": "rgb(63, 63, 63)"}}))

# Layout
fig.update_layout(title = "Casos de Covid em Junho e Julho nos EUA", showlegend = True)
# Gráfico
fig.show()

## Waterfall Múltiplo
**Objetivo**: Exibir aumento e diminuição de contagens no gráfico em cascata
Apresente o mesmo gráfico anterior para os EUA e Brasil na mesma área de plotagem.
# ->
# Já temos os dados dos EUA, agora preparamos os dados do Brasil
br_confirmed_july = covid[(covid['Country/Region']=='Brazil')&(covid['month']=='July')]
[['month','Confirmed']].groupby('month').sum().reset_index()['Confirmed']
br_confirmed_june = covid[(covid['Country/Region']=='Brazil')&(covid['month']=='June')]
[['month','Confirmed']].groupby('month').sum().reset_index()['Confirmed']
br_recovered_july = covid[(covid['Country/Region']=='Brazil')&(covid['month']=='July')]
[['month','Recovered']].groupby('month').sum().reset_index()['Recovered']
br_recovered_june = covid[(covid['Country/Region']=='Brazil')&(covid['month']=='June')]
[['month','Recovered']].groupby('month').sum().reset_index()['Recovered']
br_death_july = covid[(covid['Country/Region']=='Brazil')&(covid['month']=='July')][['month','Deaths']].groupby('month').sum().reset_index()
['Deaths']
br_death_june = covid[(covid['Country/Region']=='Brazil')&(covid['month']=='June')][['month','Deaths']].groupby('month').sum().reset_index()
['Deaths']

# ->
# Figura
fig = go.Figure()
# Adiciona a camada com o waterfall dos EUA
fig.add_trace(go.Waterfall(x = ["June",
                                "June",
                                "June",
                                "June",
                                "July",
                                "July",
                                "July",
                                "July"],
                            ["June Confirmed",
                             "June Recovered",
                             "June Death",
                             "Total Observation",
                             "July Confirmed",
                             "July Recovered",
                             "July Death",
                             "Total Observation"],
                            measure = [ "relative",
                                         "relative",
                                         "relative",
                                         "total",
                                         "relative",
                                         "relative",
                                         "relative",
                                         "total"],
                            y = [int(us_confirmed_june),
                                -int(us_recovered_june),
                                -int(us_death_june),
                                None,
                                int(us_confirmed_july),
                                -int(us_recovered_july),
                                -int(us_death_july),
                                None],
                            name = "EUA"))
# Adiciona a camada com o waterfall do Brasil
fig.add_trace(go.Waterfall(x = ["June",
                                "June",
                                "June",
                                "June",
                                "July",
                                "July",
                                "July",
                                "July"],
                            ["June Confirmed",
                             "June Recovered",
                             "June Death",
                             "Total Observation",
                             "July Confirmed",
                             "July Recovered",
                             "July Death",
                             "Total Observation"],
                            measure = [ "relative",
                                         "relative",
                                         "relative",
                                         "total",
                                         "relative",
                                         "relative",
                                         "relative",
                                         "total"],
                            y = [int(us_confirmed_june),
                                -int(us_recovered_june),
                                -int(us_death_june),
                                None,
                                int(us_confirmed_july),
                                -int(us_recovered_july),
                                -int(us_death_july),
                                None],
                            name = "Brasil"))

```

```

        "July",
        "July"],
        ["June Confirmed",
        "June Recovered",
        "June Death",
        "Total Observation",
        "July Confirmed",
        "July Recovered",

        "July Death",
        "Total Observation"]],
        measure = ["relative",
        "relative",
        "relative",
        "total",
        "relative",
        "relative",
        "relative",
        "total"],
        y = [int(br_confirmed_june),
        -int(br_recovered_june),
        -int(br_death_june),
        None,
        int(br_confirmed_july),
        -int(br_recovered_july),
        -int(br_death_july),
        None],
        name = "Brasil"))

# Layout
fig.update_layout(title = "Casos de Covid em Junho e Julho nos EUA e Brasil",
        waterfallgroupgap = 0.5,)

# Gráfico
fig.show()

# Fim

```

Gráficos Espaciais e Mapas com Plotly

```

# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Visualização de Dados e Design de Dashboards</font>
## <font color='blue'>Capítulo 2 - Métodos de Visualização</font>
## Exemplos de Gráficos Espaciais e Mapas
![title](imagens/cap02.png)
## Pacotes Python Para Manipulação de Dados e Visualização
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark

# ->
# Manipulação de Dados
import numpy as np
import pandas as pd
# Plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.colors import n_colors
from plotly.subplots import make_subplots
# Supress warnings
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")

# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions

```

```

## Carregando os Dados
# ->
# Carrega os dados
shoot = pd.read_csv('dados/police-violence-in-the-us-shootings_wash_post.csv')
shoot.head()

# ->
# Carrega os dados
terror = pd.read_csv('dados/globalterrorismdb_0718dist.csv', encoding = 'ISO-8859-1')
terror.head()

# ->
# Carrega os dados
brazil = pd.read_csv('dados/brazilian-ecommerce-olist_geolocation_dataset.csv')
brazil.head()

### Choropleth Maps
**Objetivo**: Exibir mapa de calor por contagem.
Qual o volume de mortes de pessoas negras em todos os estados dos EUA?
# ->
# Prepara os dados
black_state = shoot[shoot['race']=='B']['state'].value_counts().to_frame().reset_index().rename(columns = {'index':'state','state':'count'})

# ->
# Figura
fig = go.Figure(go.Choropleth(locations = black_state['state'],
                             z = black_state['count'].astype(float),
                             locationmode = 'USA-states',
                             colorscale = 'Reds',
                             autocolorscale = False,
                             text = black_state['state'],
                             marker_line_color = 'white',
                             colorbar_title = "Milhões USD",
                             showscale = False,))

# Layout
fig.update_layout(title_text = 'Mortes de Negros Por Estado (2015-2020)',
                  title_x = 0.5,
                  geo = dict(scope = 'usa',
                             projection = go.layout.geo.Projection(type = 'albers usa'),
                             showlakes = True,
                             lakecolor = 'rgb(255, 255, 255)'))

# Template de cores
fig.update_layout(template = "plotly_dark")
# Gráfico
fig.show()

### Mapbox com Linhas
**Objetivo**: Exibir limites e densidade por linhas usando latitude e longitude.
Como as fronteiras do comércio eletrônico são estendidas para três estados do Brasil - AC, MA e AP?
# ->
# Prepara os dados
states = brazil[(brazil['geolocation_state']=='AC')|(brazil['geolocation_state']=='MA')|(brazil['geolocation_state']=='AP')]

# ->
# Figura
fig = px.line_mapbox(states,
                    lat = "geolocation_lat",
                    lon = "geolocation_lng",
                    color = "geolocation_state",
                    zoom = 3,
                    height = 300)

# Layout
fig.update_layout(mapbox_style = "stamen-terrain",
                  mapbox_zoom = 2,
                  mapbox_center_lat = -11,
                  margin = {"r":0,"t":0,"l":0,"b":0})

# Gráfico
fig.show()

### Bubble Maps
**Objetivo**: Exibir densidade de valores com bolhas sobre latitude e longitude.
Quantas pessoas foram mortas em ataques terroristas em cada estado dos EUA?
# ->
# Prepara os dados
us_terror = terror.loc[(terror['country_txt']=='United States') & (terror['provstate']!='Unknown')][['provstate','latitude','longitude','nkill']]
map_terror = us_terror.groupby(['provstate']).agg({'nkill':'sum', 'latitude':'mean','longitude':'mean'}).reset_index()

# ->

```

```

# Parâmetros para os gráficos
limits = [(0,5), (6,15), (16,3000)]
colors = ["royalblue", "crimson", "lightseagreen"]
cities = []
scale = 1

# ->
# Figura
fig = go.Figure()
# Loop para configuração dos parâmetros associados aos dados
for i in range(len(limits)):
    lim = limits[i]
    df_sub = map_terror[lim[0]:lim[1]]
    fig.add_trace(go.Scattergeo(locationmode = 'USA-states',
                                lon = df_sub['longitude'],
                                lat = df_sub['latitude'],
                                marker = dict(size = df_sub['nkill'] / scale,
                                                color = colors[i],
                                                line_color = 'rgb(40,40,40)',
                                                line_width = 0.5,
                                                sizemode = 'area'),
                                name = '{0} - {1}'.format(lim[0],lim[1])))
fig.update_layout(title_text = 'Ataques Terroristas nos EUA',
                  title_x = 0.5,
                  showlegend = True,
                  geo = dict(scope = 'usa',
                              landcolor = 'rgb(217, 217, 217)'))

# Gráfico
fig.show()

### Mapbox Density
**Objetivo**: Exibir densidade de valores com mapa de calor sobre latitude e longitude.
Quantas pessoas foram mortas em ataques terroristas em todo o mundo?
# ->
# Prepara os dados
all_terror = terror[['city','latitude','longitude','nkill']]
map_all_terror = all_terror.groupby(['latitude','longitude']).agg({'nkill':'sum'}).reset_index()

# ->
# Mapbox
fig = px.density_mapbox(map_all_terror,
                        lat = 'latitude',
                        lon = 'longitude',
                        z = 'nkill',
                        radius = 10,
                        center = dict(lat = 31, lon = 36), zoom = 1, mapbox_style = "stamen-terrain")

# Layout
fig.update_layout(title_text = 'Pessoas Mortas em Ataques Terroristas ao Redor do Mundo',
                  title_x = 0.5,
                  showlegend = True)

# Gráfico
fig.show()

### Mapbox Layers
**Objetivo**: Exibir pontos de latitude e longitude.
Qual o total de lojas nos estados RO, AM, AC, AP, RR no Brasil?
# ->
# Prepara os dados
brazil_5 = brazil[(brazil['geolocation_state']=='RO')|(brazil['geolocation_state']=='AM')|(brazil['geolocation_state']=='AC')|
(brazil['geolocation_state']=='AP')|(brazil['geolocation_state']=='RR')]

# ->
# Figura
fig = px.scatter_mapbox(brazil_5,
                        lat = "geolocation_lat",
                        lon = "geolocation_lng",
                        hover_name = "geolocation_city",
                        hover_data = ["geolocation_state"],
                        color_discrete_sequence = ["fuchsia"],
                        zoom = 2,
                        center = dict(lat = -18, lon = -52),
                        height = 300)

# Layout
fig.update_layout(mapbox_style = "open-street-map")
fig.update_layout(margin = {"r":0,"t":0,"l":0,"b":0})
# Gráfico
fig.show()

```

```
# Fim
```

Gráficos Estatísticos com Plotly

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Visualização de Dados e Design de Dashboards</font>
## <font color='blue'>Capítulo 2 - Métodos de Visualização</font>
## Exemplos de Plots Estatísticos
![title](imagens/cap02.png)
## Pacotes Python Para Manipulação de Dados e Visualização
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark

# ->
# Manipulação de Dados
import numpy as np
import pandas as pd
# Plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.colors import n_colors
from plotly.subplots import make_subplots

# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions

## Carregando os Dados
# ->
# Carrega os dados
campus = pd.read_csv('dados/factors-affecting-campus-placement-Placement_Data_Full_Class.csv')
campus.head()

# ->
# Carrega os dados
house = pd.read_csv("dados/house-prices-advanced-regression-techniques-train.csv")
house.head()

# ->
# Carrega os dados
world = pd.read_csv('dados/world-university-rankings-cwurData.csv')
world.head()

# ->
# Carrega os dados
google = pd.read_csv("dados/google-play-store-apps-googleplaystore.csv")
google.head()

### Histograma
**Objetivo**: Exibir distribuição de uma variável contínua.
Qual é a distribuição salarial dos graduados em Computação?
# ->
# Prepara os dados
campus_computer = campus[campus['degree_t']=='Comm&Mgmt'].dropna()['salary']

# ->
# Figura
fig = go.Figure(data = [go.Histogram(x = campus_computer,
                                   marker_color = "orange",
                                   xbins = dict(start = 200000, end = 1000000, size = 10000))])

# Layout
fig.update_layout(title = "Distribuição de Salários Para Graduados em Computação",
                  xaxis_title = "Salário",
                  yaxis_title = "Frequência")

# Gráfico
fig.show()
```



```

#### Histograma Normalizado
**Objetivo**: Exibir distribuição de uma variável contínua.
Qual a distribuição salarial dos graduados em Ciência e Tecnologia de forma normalizada?
# ->
# Prepara os dados
campus_science = campus[campus['degree_t']=='Sci&Tech']['salary']

# ->
# Figura
fig = go.Figure(data = [go.Histogram(x = campus_science,
                                   histnorm = 'probability',
                                   marker_color = "magenta")])

# Layout
fig.update_layout(title = "Distribuição de Salários Para Graduados em Ciências",
                  xaxis_title = "Salário",
                  yaxis_title = "Frequência")

# Gráfico
fig.show()

#### Histogramas Sobrepostos
**Objetivo**: Exibir a distribuição de uma variável contínua para diferentes grupos.
Qual é a distribuição percentual de graduados em Computação e Ciências de forma sobreposta?
# ->
# Prepara os dados
per_com = campus[campus['degree_t']=='Comm&Mgmt']['degree_p']
per_sci = campus[campus['degree_t']=='Sci&Tech']['degree_p']

# ->
# Figura
fig = go.Figure()
# Adiciona os historamas
fig.add_trace(go.Histogram(x = per_com, marker_color = "green", name = "Graduados em Computação"))
fig.add_trace(go.Histogram(x = per_sci, marker_color = "orange", name = "Graduados em Ciências"))
# Sobreposição dos histogramas
fig.update_layout(barmode = 'overlay')
# Reduz a opacidade dos histogramas
fig.update_traces(opacity = 0.74)
# Layout
fig.update_layout(title = "Distribuição da Porcentagem de Graduados em Computação e Ciências",
                  xaxis_title = "Percentual",
                  yaxis_title = "Frequência")

# Gráfico
fig.show()

#### Histogramas Empilhados
**Objetivo**: Exibir a distribuição de uma variável contínua para diferentes grupos.
Mesmo gráfico anterior, mas agora em formato empilhado.
# ->
# Figura
fig = go.Figure()
# Histogramas
fig.add_trace(go.Histogram(x = per_com, marker_color = "green", name = "Graduados em Computação"))
fig.add_trace(go.Histogram(x = per_sci, marker_color = "orange", name = "Graduados em Ciências"))
# Stack
fig.update_layout(barmode='stack')
# Redução da opacidade
fig.update_traces(opacity = 0.73)
# Layout
fig.update_layout(title = "Distribuição da Porcentagem de Graduados em Computação e Ciências",
                  xaxis_title = "Percentual",
                  yaxis_title = "Frequência")

# Gráfico
fig.show()

#### Distplot
**Objetivo**: Exibir distribuição de uma variável contínua.
Qual é a distribuição de preços para casa com nota de avaliação igual a 4?
# ->
# Preparação dos dados
class_1 = house[house['OverallCond']==4]['SalePrice']
# Dados para o histograma
hist_data = [class_1]
# Labels
group_labels = ['Distribuição de Preços Para Casas com Avaliação 4']
# Cores
colors = ['green']

```

```

# ->
# Figura
fig = ff.create_distplot(hist_data, group_labels, colors = colors, bin_size = [10000])
# Gráfico
fig.show()

#### Distplot Múltiplo
**Objetivo**: Exibir distribuição de uma variável contínua para várias categorias.
Mesmo gráfico anterior para as avaliações 4, 5 e 6.
# ->
# Preparação dos dados
class_1 = house[house['OverallCond']==4]['SalePrice']
class_2 = house[house['OverallCond']==5]['SalePrice']
class_3 = house[house['OverallCond']==6]['SalePrice']

# ->
# Dados para cada histograma
hist_data = [class_1, class_2, class_3]
# Labels
group_labels = ['Distribuição de Preços Para Casas com Avaliação 4',
                'Distribuição de Preços Para Casas com Avaliação 5',
                'Distribuição de Preços Para Casas com Avaliação 6']
# Cores
colors = ['blue', "green", "magenta"]

# ->
# Figura
fig = ff.create_distplot(hist_data, group_labels, colors = colors, bin_size = [10000,10000,10000])
# Gráfico
fig.show()

#### Distplot Curve
**Objetivo**: Exibe a distribuição de uma variável contínua para várias categorias com curva em vez de barra.
Mesmo gráfico anterior somente com as linhas.
# ->
# Dados para cada histograma
hist_data = [class_1, class_2, class_3]
# Labels
group_labels = ['Distribuição de Preços Para Casas com Avaliação 4',
                'Distribuição de Preços Para Casas com Avaliação 5',
                'Distribuição de Preços Para Casas com Avaliação 6']
# Cores
colors = ['blue', "green", "magenta"]

# ->
# Figura
fig = ff.create_distplot(hist_data, group_labels, show_hist = False, colors = colors, bin_size = [10000,10000,10000])
# Gráfico
fig.show()

#### Density Contour
**Objetivo**: Exibe 2 histogramas e um scatter plot na mesma área de plotagem.
Relação das variáveis tamanho do lote e preço da casa.
# ->
# Prepara os dados
cond_8 = house[house['OverallQual']==8]
# x e y
x = cond_8['LotArea']
y = cond_8['SalePrice']

# ->
# Figura
fig = go.Figure()
# Countour
fig.add_trace(go.Histogram2dContour(x = x,
                                   y = y,
                                   colorscale = 'gray',
                                   reversescale = True,
                                   xaxis = 'x',
                                   yaxis = 'y'))
# Scatter Plot
fig.add_trace(go.Scatter(x = x,
                        y = y,
                        xaxis = 'x',
                        yaxis = 'y',
                        mode = 'markers',
                        marker = dict(color = "red", size = 3)))
# Histograma

```

```

fig.add_trace(go.Histogram(y = y,
                           xaxis = 'x2',
                           marker = dict(color = "blue")))
# Histograma
fig.add_trace(go.Histogram(x = x,
                           yaxis = 'y2',
                           marker = dict(color = "blue")))
# Layout
fig.update_layout(autosize = False,
                  xaxis = dict(zeroline = False, domain = [0,0.85], showgrid = False),
                  yaxis = dict(zeroline = False, domain = [0,0.85], showgrid = False),
                  xaxis2 = dict(zeroline = False, domain = [0.85,1], showgrid = False),
                  yaxis2 = dict(zeroline = False, domain = [0.85,1], showgrid = False),
                  height = 600,
                  width = 600,
                  bargap = 0,
                  hovermode = 'closest',
                  showlegend = False,
                  title_text = "Density Contour e Tamanho e Preço Para Casas de Avaliação Igual a 8", title_x = 0.5)
# Gráfico
fig.show()

### Box Plot
**Objetivo**: Exibir distribuição de uma variável contínua.
Como a pontuação é distribuída para diferentes universidades na Alemanha?
# ->
# Prepara os dados
germany_score = world[world['country']=="Germany"]['score']

# ->
# Figura
fig = go.Figure(go.Box(y = germany_score, name = "Pontuação"))
# Layout
fig.update_layout(title = "Distribuição de Pontuação das Universidades da Alemanha")
# Gráfico
fig.show()

### Box Plot Agrupado
**Objetivo**: Exibe a distribuição de uma variável contínua para dois ou mais grupos.
Compare as avaliações das universidades no Canadá e no Brasil.
# ->
# Prepara os dados
score_canada = world[world['country']=="Canada"]['score']
score_brazil = world[world['country']=="Brazil"]['score']

# ->
# Figura
fig = go.Figure()
# Adiciona o box plot do Canadá
fig.add_trace(go.Box(y = score_canada, marker_color = "red", name = "Avaliação das Universidades do Canadá"))
# Adiciona o box plot do Brasil
fig.add_trace(go.Box(y = score_brazil, marker_color = "green", name = "Avaliação das Universidades do Brasil"))
# Layout
fig.update_layout(title = "Distribuição das Avaliações das Universidades no Canadá e no Brasil")
# Gráfico
fig.show()

### Box Plot com Média e Desvio Padrão
**Objetivo**: Exibir a distribuição de uma variável contínua
para dois ou mais grupos com média e desvio padrão.
Como é a distribuição de classificação para as categorias de aplicativos da Play Store?
# ->
# Prepara os dados
rating_maps = google[google['Category']=="MAPS_AND_NAVIGATION"]['Rating']
rating_life = google[google['Category']=="LIFESTYLE"]['Rating']

# ->
# Figura
fig = go.Figure()
# Adiciona o primeiro box plot
fig.add_trace(go.Box(y = rating_maps,
                    boxmean = 'sd',
                    marker_color = "darkorchid",
                    name = "Avaliação de Apps de Mapas"))
# Adiciona o segundo box plot
fig.add_trace(go.Box(y = rating_life,
                    boxmean = 'sd',
                    marker_color = "blue",

```

```

        name = "Avaliação de Apps Sobre Lifestyle"))

# Layout
fig.update_layout(title = "Distribuição de Avaliações de Apps da Google Playstore")
# Gráfico
fig.show()

#### Scatter Plot
**Objetivo**: Relação entre valores numéricos.
Quanta dependência existe entre Tamanho do Lote e Preço da Casa?
# ->
# Figura
fig = px.scatter(house, x = 'LotArea', y = 'SalePrice')
# Layout
fig.update_layout(title = 'Tamanho do Lote x Preço', xaxis_title = "Tamanho do Lote", yaxis_title = "Preço")
# Gráfico
fig.show()

#### Scatter Plot Categorizado
**Objetivo**: Relação entre valores numéricos com um campo categórico.
Quanta dependência existe entre Tamanho do Lote e Preço com o Shape da Casa?
# ->
# Figura
fig = px.scatter(house, x = 'LotArea', y = 'SalePrice', color = 'LotShape')
# Layout
fig.update_layout(title = 'Tamanho do Lote x Preço x Shape', xaxis_title = "Tamanho do Lote", yaxis_title = "Preço")
# Gráfico
fig.show()

#### Scatter Plot Customizado
**Objetivo**: Relação entre valores numéricos com adição de categorização por um campo e aumento do tamanho do ponto de dados por outro campo numérico.
Quanta dependência existe entre a qualidade da educação e a pontuação dos alunos em diferentes países com base no número de alunos?
# ->
# Figura
fig = px.scatter(world, x = "quality_of_education", y = "score", color = "country", size = "citations")
# Layout
fig.update_layout(title = 'Qualidade da Educação x Pontuação x Número de Alunos',
                    xaxis_title = "Qualidade da Educação",
                    yaxis_title = "Pontuação",
                    template = 'seaborn')
# Gráfico
fig.show()

# Fim

```

Heatmaps e Wordclouds com Plotly

```

# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Visualização de Dados e Design de Dashboards</font>
## <font color='blue'>Capítulo 2 - Métodos de Visualização</font>
## Exemplos de Heatmaps e Wordcloud
![title](imagens/cap02.png)
## Pacotes Python Para Manipulação de Dados e Visualização
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark

# ->
# Pacote wordcloud
!pip install -q wordcloud

# ->
# Manipulação de Dados
import numpy as np
import pandas as pd
import datetime
from datetime import date, timedelta
# Plotly
import plotly.express as px

```

```

import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.colors import n_colors
from plotly.subplots import make_subplots
# Wordcloud
import wordcloud
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions

## Carregando os Dados
# ->
# Carrega os dados
titanic = pd.read_csv('dados/titanic-train.csv')
titanic

# ->
# Carrega os dados
covid = pd.read_csv('dados/novel-corona-virus-2019-dataset-covid_19_data.csv')
covid

# ->
# Carrega os dados
world = pd.read_csv('dados/world-university-rankings-cwurData.csv')
world

#### Heatmap
**Objetivo** : Exibir mapa de calor de variáveis quantitativas com uma variável numérica densa.
Como variam os preços dos bilhetes para todos os passageiros de diferentes sexos e portos de embarque no Titanic?
# ->
# Figura
fig = go.Figure(data = go.Heatmap(z = titanic['Fare'],
                                x = titanic['Sex'],
                                y = titanic['Embarked'],
                                hoverongaps = False))

# Layout
fig.update_layout(title = 'Heatmap Preço do Ticket x Gênero de Passageiros do Titanic Por Porto de Embarque',
                  xaxis_title = "Gênero",
                  yaxis_title = "Porto de Embarque")

# Gráfico
fig.show()

#### Heatmap Date Axis
**Objetivo** : Exibir mapa de calor de várias séries temporais.
Quantas mortes suspeitas de Covid ocorreram no Brasil, EUA e Canada no mês anterior?
# ->
# Prepara os dados
covid['ObservationDate'] = pd.to_datetime(covid['ObservationDate'])
days_before = (date.today() - timedelta(days = 360)).isoformat()
last_month_df = covid[(covid['ObservationDate'] > days_before) & ((covid['Country/Region']=='US') | (covid['Country/Region']=='Brazil') |
(covid['Country/Region']=='Canada'))]
country = last_month_df['Country/Region'].unique()
dates = pd.to_datetime(last_month_df['ObservationDate'].unique())
num_deaths=[]
for i in country:
    num_deaths.append(last_month_df[last_month_df['Country/Region']==i]
[['ObservationDate','Deaths']].groupby('ObservationDate').sum().reset_index()['Deaths'])

# ->
# Figura
fig = go.Figure(data = go.Heatmap(z = num_deaths,
                                x = dates,
                                y = country,
                                colorscale = 'Viridis'))

# Layout
fig.update_layout(title = 'Mortes Por Suspeita de Covid nos Últimos 30 Dias', xaxis_nticks = 30)

# Gráfico
fig.show()

#### Imshow
**Objetivo** : Exibir mapa de calor de variáveis quantitativas com uma variável numérica como densa (semelhante ao heatmap).
Qual a variação média dos preços dos bilhetes para passageiros de diferentes sexos e seu porto de embarque?
# ->
# Prepara os dados

```

```

emb_male = []
for i in titanic['Embarked'].unique():
    emb_male.append(titanic[(titanic['Sex']=='male') & (titanic['Embarked']==i)]['Fare'].mean())
emb_female = []
for j in titanic['Embarked'].unique():
    emb_female.append(titanic[(titanic['Sex']=='female') & (titanic['Embarked']==j)]['Fare'].mean())
emb = [emb_male, emb_female]

# ->
# Figura
fig = px.imshow(emb,
                 labels = dict(x = "Porto de Embarque", y = "Gênero", color = "Valor Médio do Ticket"),
                 x = titanic['Embarked'].unique(),
                 y = titanic['Sex'].unique())
# Eixos
fig.update_xaxes(side = "top")
# Gráfico
fig.show()

### Wordcloud
**Objetivo** : Exibir as palavras de uma coluna com o tamanho representando a frequência.
Quais países tem mais universidades avaliadas no dataset de avaliações de universidades?
# ->
# Colunas
world.columns

# ->
# Extraí a coluna com os países
df = world.country
df.head(10)

# ->
# Figura
plt.subplots(figsize = (8,8))
# Wordcloud
wordcloud = WordCloud(background_color = 'white',
                       width = 512,
                       height = 384).generate(' '.join(df))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()

# Fim

```

Sunburst Diagram e Candlesticks com Plotly

```

# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Visualização de Dados e Design de Dashboards</font>
## <font color='blue'>Capítulo 2 - Métodos de Visualização</font>
## Sunburst Diagram e Candlesticks com Plotly
![title](imagens/cap02.png)
## Pacotes Python Para Manipulação de Dados e Visualização
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark

# ->
# Manipulação de Dados
import numpy as np
import pandas as pd
import datetime
from datetime import date, timedelta
# Plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.colors import n_colors
from plotly.subplots import make_subplots

```



```

        title = "Percentual de Candidatos Empregados e Não Empregados")
# Gráfico
fig.show()

#### 3D Scatter Colorido
**Objetivo**: Exibir relação entre 3 variáveis com forma e cor diferentes.
Mesmo gráfico anterior com shapes e cores diferentes para as variáveis.
# ->
# Figura
fig = px.scatter_3d(campus,
                    x = 'ssc_p',
                    y = 'hsc_p',
                    z = 'degree_p',
                    color = 'etest_p',
                    size = 'etest_p',
                    size_max = 18,
                    symbol = 'status',
                    opacity = 0.7)

# Layout
fig.update_layout(margin = dict(l = 0, r = 0, b = 0, t = 0))
# Gráfico
fig.show()

#### 3D Surface
**Objetivo**: Exibir relação de superfície entre diversos valores.
Como as pontuações estão relacionadas entre as universidades dos 20 principais países?
# ->
# Prepara os dados
top_countries = world['country'].value_counts()[:20].reset_index()['index']
score = []
for i in top_countries:
    score.append(list(world[world['country']==i]['score']))
z_data = np.array(score)

# ->
# Figura
fig = go.Figure(data = [go.Surface(z = z_data)])
# Layout
fig.update_layout(title = 'Pontuação das Universidades dos 20 Principais Países',
                  autosize = False,
                  width = 500,
                  height = 500,
                  margin = dict(l = 65, r = 50, b = 65, t = 90))
# Gráfico
fig.show()

#### 3D Line
**Objetivo**: Exibir relação entre 3 variáveis em linhas.
Como se relacionam a população, o PIB e o ano nos países europeus?
# ->
# Prepara os dados
df = px.data.gapminder().query("continent=='Europe'")

# ->
# Figura
fig = px.line_3d(df, x = "gdpPercap", y = "pop", z = "year", color = 'country')
# Layout
fig.update_layout(title_text = 'População, PIB e Ano na Europa', title_x = 0.5)
# Gráfico
fig.show()

# Fim

```

7 Dicas Para Melhorar a Leitura e Entendimento de Gráficos

1. Se um gráfico requer muita interpretação ele está ruim
2. É mais fácil entender comprimento do que área
3. Gráficos 3D podem distorcer os dados
4. Informações visuais devem ser apresentadas primeiro

5. A imagem importa, mas as letras também
 1. Consistente: formatar do mesmo modo elementos semelhantes do gráfico
 2. Legível: variações no tipo, cor e tamanho da letra podem ajudar a leitura
6. Tipografia e orientação do texto são importantes
7. Informações visuais também precisam de uma pausa para respirar

Infográficos

Um infográfico é caracterizado por ilustrações explicativas sobre um tema ou assunto. Infográfico é a junção das palavras info (informação) e gráfico (desenho, imagem, representação visual), ou seja, um infográfico é um desenho ou imagem que, com o auxílio de um texto, explica ou informa sobre um assunto que não seria muito bem compreendido somente com um texto. Os infográficos são muito utilizados em jornais, mapas, manuais técnicos, educativos e científicos, e também em web sites.

Infográficos são representações visuais de informação. Esses gráficos são usados onde a informação precisa ser explicada de forma mais dinâmica, como em mapas, jornalismo e manuais técnicos, educativos ou científicos. Pode utilizar a combinação de fotografia, desenho e texto. Um exemplo de infográfico do tipo mais simples poderia ser uma linha de tempo onde ao selecionar determinados períodos aparecem imagem e textos explicativos.

Por unir texto e imagens, o infográfico atua em duas zonas distintas do cérebro humano: o lado direito, responsável por entender e interpretar figuras; e o lado esquerdo, que é focado na escrita e no raciocínio lógico. Assim, os infográficos acabam por simplificar a interpretação dos conteúdos, pois as duas áreas do cérebro atuam em conjunto.

A vantagem do infográfico é a sua capacidade de ser personalizado visualmente de acordo com o tipo de público-alvo a ser abordado, tema, área de atuação, entre outros. Aliás, por ser uma ferramenta visual, a apresentação do infográfico é muito importante para a sua correta interpretação. Um bom infográfico deve ser apresentado de modo organizado, facilitando a compreensão das informações nele contidas por diferentes níveis de pessoas, desde os técnicos até os leigos no assunto, por exemplo.

As representações gráficas como ferramentas para explicar ideias, conceitos e processos são utilizadas desde tempos pré-históricos. No entanto, os infográficos como são conhecidos na atualidade só começaram a serem feitos no princípio do século XVI. Leonardo da Vinci, por exemplo, utilizou vastamente desenhos e figuras para explicar de modo mais simplificado os seus estudos e teorias, sobre diversos assuntos, mas em especial a respeito da anatomia humana.

Embora os infográficos não sejam explorados aqui no curso, considere utilizá-los como forma de consolidar suas visualizações e storytelling em um mesmo objeto. No site visual capitalist você encontra muitos exemplos de infográficos sobre os mais variados assuntos.

<http://www.visualcapitalist.com>

Quizz

Um gráfico simplifica bastante a compreensão. A visualização de dados é importante, porque facilita a legibilidade das informações. Isso significa que, em vez de apresentar enormes blocos de texto com diversos números e tabelas, você garantirá a compreensão do interlocutor ao apresentar gráficos que ajudem a transmitir a mensagem desejada sem dar espaço para mal-entendidos.

Justamente por identificar padrões e relações que não seriam vistos de outra maneira é que a utilização de gráficos é responsável por ajudar na percepção de necessidades de otimização.

A visualização de dados é a representação dos mesmos em um formato gráfico. O objetivo da visualização é simplificar o valor dos dados, promover a compreensão sobre eles, e comunicar conceitos e ideias importantes.

Diagrama NÃO é um exemplo de visual encoding.

Exemplos de visual encoding: tamanho, textura e orientação.

O gráfico de colunas é um gráfico simples, que serve para criar distribuições de valores diferentes em série. **(FALSO!!!)**

Tipos de Gráficos e Sua Funcionalidade

- Gráfico de Barras
 - Legibilidade: fácil
 - Função: comparação
 - Comparação entre muitos itens
 - Exemplos: Visitas, vendas, investimento
- Gráfico de Colunas
 - Legibilidade: fácil
 - Função: comparação
 - Dados em poucos períodos e poucas categorias ou comparação entre poucos itens
 - Exemplos: frequência relativa, contagem, percentuais
- Gráfico de Linhas
 - Legibilidade: fácil
 - Função: comparação
 - Dados em muitos períodos e não cíclicos ou dados em poucos períodos e muitas categorias
 - Exemplos: visitas, vendas
- Gráfico de Pizza
 - Legibilidade: fácil
 - Função: composição
 - Composição estática, que mostre uma porção simples do total

- Exemplos: visitantes novos e retornantes
- Gráfico de Rosca (Donut)
 - Legibilidade: média
 - Função: composição
 - Composição estática, que mostre porções do total
 - Exemplos: visitantes novos e retornantes
- Gráfico de Área
 - Legibilidade: média
 - Função: composição
 - Composição variável ao longo do tempo, com muitos períodos ou tipos
 - Exemplos: visitas, vendas, receita
- Gráfico de Área Empilhada (Stacked Area)
 - Legibilidade: média
 - Função: composição
 - Composição variável ao longo do tempo, com muitos períodos ou tipos
 - Exemplos: visitas, vendas, receita
- Gráfico de Bolhas (Bubble Plot)
 - Legibilidade: difícil
 - Função: relação/comparação
 - Interdependência entre variáveis
 - Exemplos: contagem, frequência e proporção, cliques e conversões
- Gráfico de Radar (Radar Plot)
 - Legibilidade: difícil
 - Função: comparação
 - Dados em períodos cíclicos ou muitas variáveis no mesmo eixo ou avaliação qualitativa segundo critérios que podem ser quantificados
 - Exemplos: avaliação de ferramentas, alinhamento dos esforços de mídia com os princípios da empresa, dados complementares a um mapa
- Gráfico de Cachoeira (Waterfall Plot)
 - Legibilidade: média
 - Função: composição
 - Composição estática que mostre acumulação e subtração do total

- Exemplos: controle de estoque, fluxo de caixa
- Gráfico Packed Circle Diagram
 - Legibilidade: difícil
 - Função: composição
 - Composição estática que mostre acumulação e subtração do total
 - Exemplos: controle de estoque, fluxo de caixa, visualização de hierarquia
- Gráfico Histograma
 - Legibilidade: média
 - Função: distribuição
 - Distribuição de frequências ou densidade de probabilidades, com variável única
 - Exemplos: exposição de fotografia (distribuição de pixels claros e escuros), distribuição de renda
- Gráfico Density Plots
 - Um gráfico de densidade permite visualizar a distribuição de dados em um intervalo contínuo ou período de tempo
- Gráfico de Dispersão (Scatter Plots)
 - Legibilidade: média
 - Função: comparação ou relação
 - Relação de duas variáveis ou Interdependência entre duas variáveis
 - Exemplos: investimento em marketing e vendas
 - Pode apontar: Nenhuma correlação, Possível correlação positiva ou negativa, Correlação positiva ou negativa
 - Correlação não é causalidade!
- Gráfico Matriz de Gráficos de Dispersão (Scatter Plots Matrix)
 - Legibilidade: média
 - Função: comparação ou relação
 - Relação de duas variáveis ou Interdependência entre duas variáveis
 - Exemplos: investimento em marketing e vendas
- Gráfico Box e Whisker Plot
 - Um Box e Whisker Plot (ou apenas Box Plot ou ainda Gráfico de Caixa) é uma maneira conveniente de exibir visualmente grupos de dados numéricos através de seus quartis.
 - Tipicamente usado em estatísticas descritivas, os gráficos de caixas são uma ótima maneira de examinar rapidamente as características de uma variável sendo, portanto, usado para análise univariada.

- Gráfico Heatmaps
 - Heatmaps são visualizações de dados através de variações na coloração do diagrama.
 - Pode ser utilizado para dados numéricos e categóricos.
- Gráfico Word Clouds (Nuvem de Palavras)
 - Nuvem de palavras é um método de visualização que exibe a frequência com que as palavras aparecem em um dado corpo de texto, fazendo o tamanho de cada palavra proporcional à sua frequência.
 - A cor usada nas nuvens de palavras geralmente não tem sentido e é essencialmente estética, mas pode ser usada para categorizar palavras ou para exibir outra variável de dados.
- Gráfico Sunburst Diagram
 - Este tipo de visualização mostra hierarquia através de uma série de anéis, que são cortados para cada nó de categoria.
- Gráfico Network Diagram
 - Através do mapeamento de sistemas conectados, os Diagramas de Rede podem ser usados para interpretar a estrutura de uma rede.
 - Este tipo de visualização mostra como os dados são interligados através da utilização de nós/vértices e linhas para representar as ligações e ajudar a identificar o tipo de relação entre grupos de entidades.
- Gráfico Candlesticks
 - Os gráficos de Candlestick se originaram no Japão mais de 100 anos antes do Ocidente desenvolver os gráficos de barras. Nos anos 1700, um japonês chamado Homma descobriu que, embora houvesse uma relação entre o preço e a oferta e demanda de arroz, os mercados eram fortemente influenciados pelas emoções dos comerciantes.
 - Os Candlesticks (castiçais) mostram essa emoção, representando visualmente o tamanho dos movimentos de preços com cores diferentes. Os comerciantes usam os Candlesticks para tomar decisões de negociação com base em padrões que ocorrem regularmente que ajudar a prever a direção de curto prazo do preço.
 - Assim como um gráfico de barras, um candlestick diário mostra os preços de abertura, alta, baixa e fechamento do mercado no dia. O candlestick tem uma parte larga que é chamada de “corpo real”. Este corpo real representa a faixa de preço entre a abertura e o fechamento do dia de negociação. Quando o corpo real está preenchido ou preto, significa que o fechamento foi menor do que a abertura. Se o corpo real está vazio, significa que o fechamento foi maior do que o valor de abertura.

9.3. Organização Visual

Qual o Objetivo de Uma Visualização?

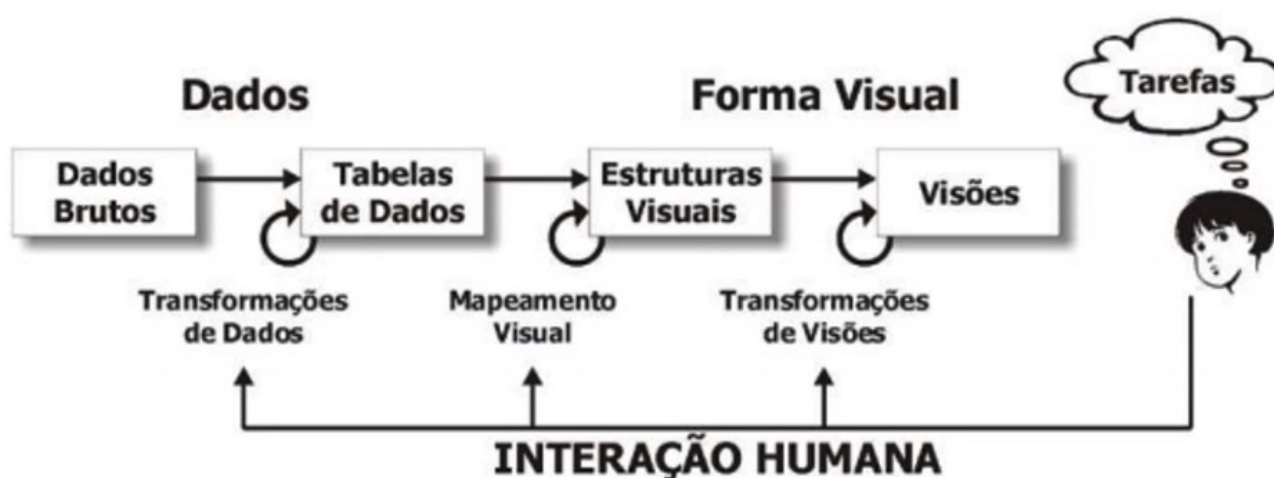
O objetivo básico e principal de qualquer visualização é sempre o mesmo: a partir de um grande volume de dados extrair o máximo de informação de uma forma rápida, clara e precisa.

Visualização de Informação x Visualização Científica

Visualização de informação é aquela onde você vai transmitir uma informação para a sua audiência. É o resultado final, a entrega do processo de análise.

Já a visualização científica é normalmente utilizada durante o processo de análise.

Independentemente do tipo de visualização, o processo é basicamente o mesmo:



Identidade Visual x Design Gráfico

Identidade visual é o conjunto de elementos visuais que comunicam e representam os valores, posicionamento e diferenciais competitivos de uma empresa ou organização.

A identidade visual corporativa está relacionada com as frases, as imagens, as tipografias, os logos que comunicam a missão, visão e valores da empresa. E isso deve estar refletido na visualização de dados que você construir, seja um gráfico, infográfico ou dashboard (isso faz muita diferença!).

O Design Gráfico é a forma como você cria os componentes da identidade visual, como as cores serão combinadas, qual a posição do logo no seu gráfico, qual a geometria dos objetos.

Qual a Diferença Entre UI Design e UX Design?

UI Design ou User Interface Design (Design de Interface do Usuário) é o meio pelo qual uma pessoa interage e controla um dispositivo, software ou aplicativo.

UX Design ou User Experience Design (Design de Experiência do Usuário) pode ser resumido com um forma de englobar todos os aspectos da interação do usuário final com a empresa, seus serviços e seus produtos, ou seja, é responsável por estudar as melhores maneiras de atender as necessidades dos usuários e deixá-los satisfeitos com todo o processo.

O Cientista de Dados pode realizar um Teste A/B para análise estatística de duas interfaces diferentes e concluir qual apresentou melhor resultado de interação com os usuários.

Visualização de Dados Multidimensionais

Técnicas de Visualização de Dados Multidimensionais:

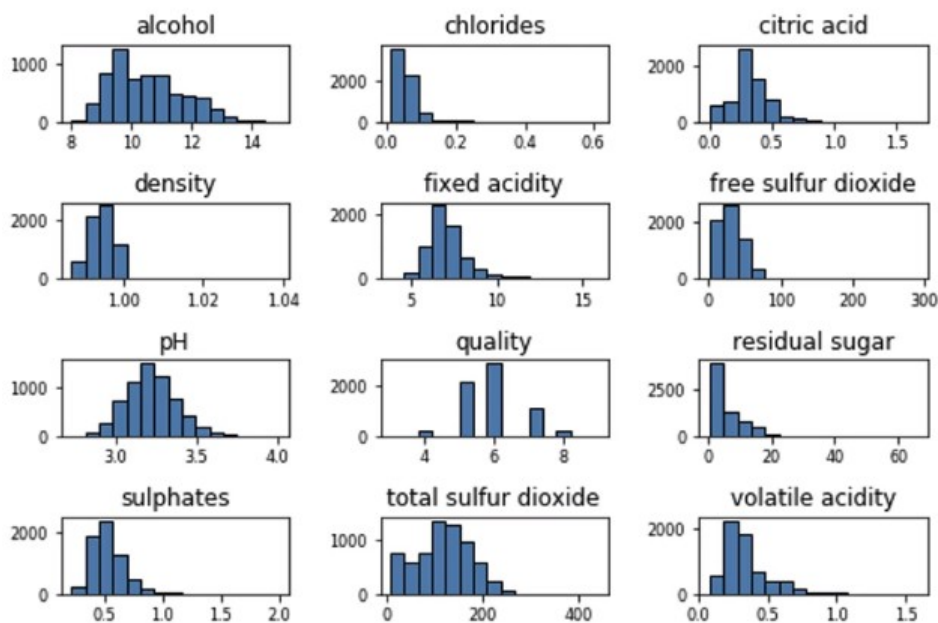
- Projeções Geométricas
- Orientadas a Pixels
- Iconográficas
- Hierárquicas
- Baseadas em Grafos
- Híbridas

Projeções Geométricas

Em projeções geométricas o objetivo é identificar projeções de interesse em conjuntos de dados multidimensionais.

Análise Univariada:

Cada gráfico representa uma variável (uma única dimensão).

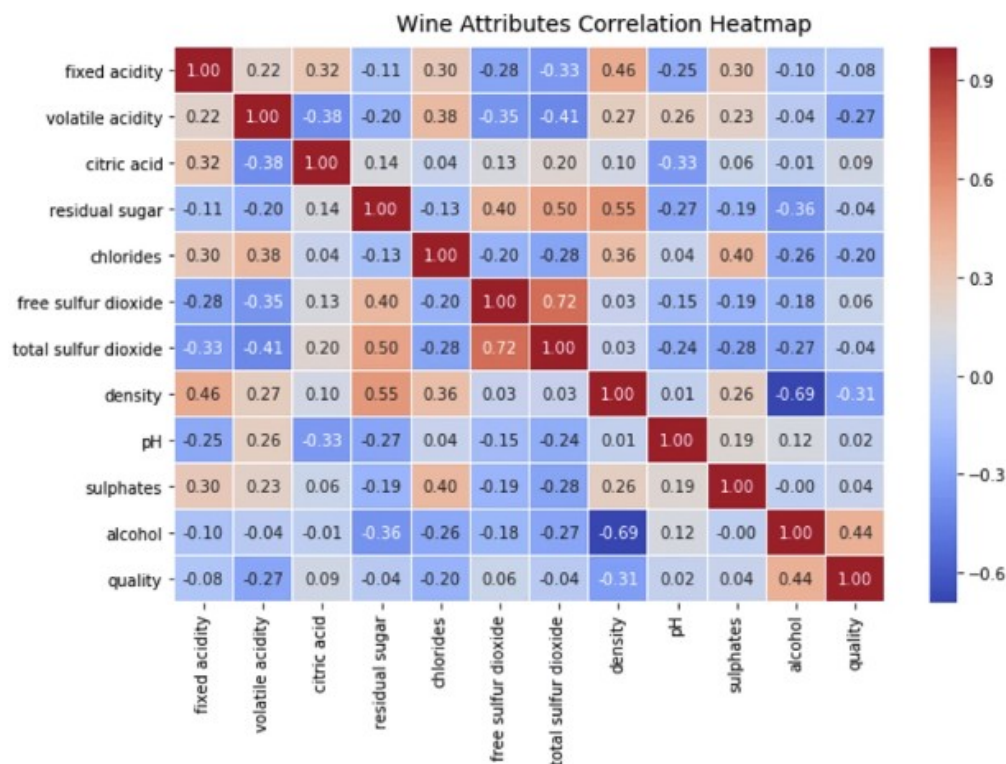


Valores possíveis (x) x Contagem (y)

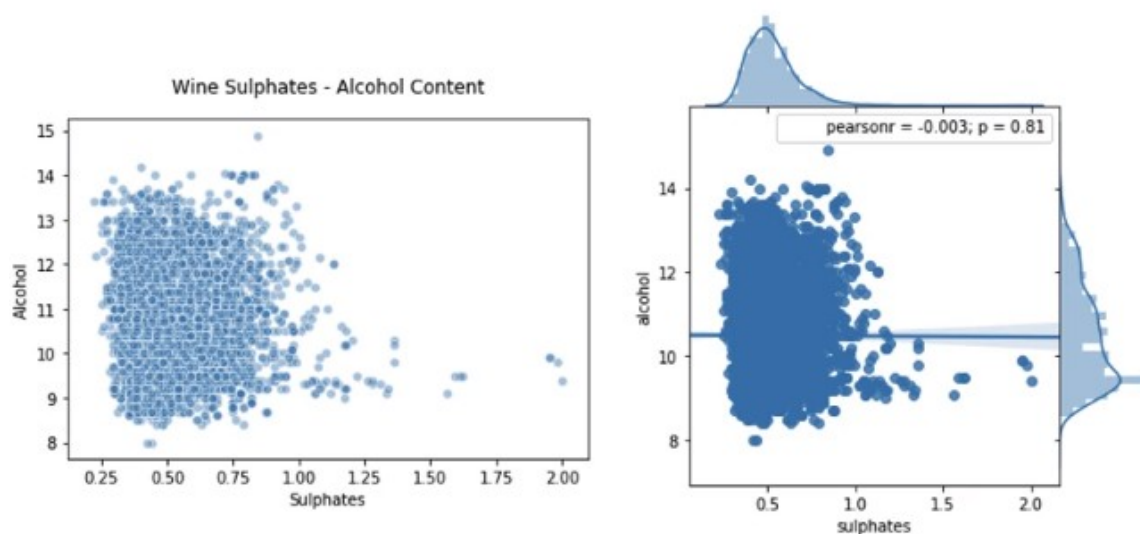
Histogramas e Gráficos de Densidade são amplamente usados nesse caso e embora tenhamos apenas uma dimensão dos dados, os gráficos apresentam uma orientação bidimensional.

Análise Multivariada – 2 Dimensões:

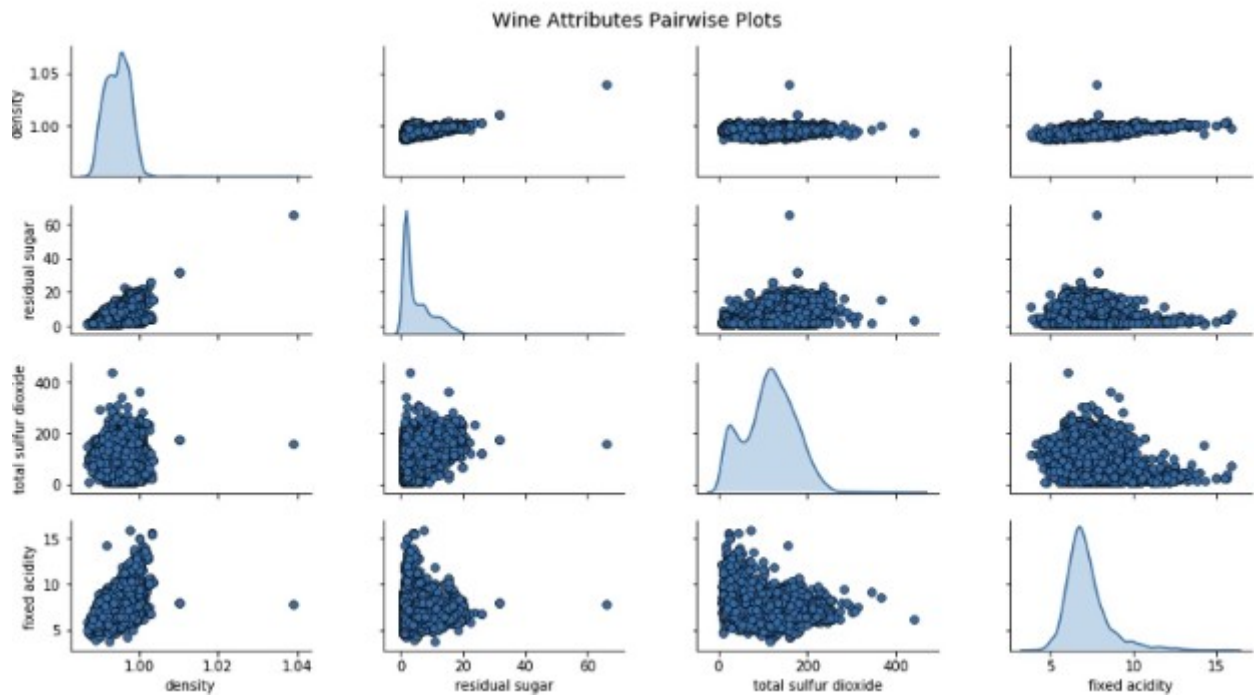
A matriz de correlação é um exemplo de gráfico para análise multivariada. A projeção geométrica é bidimensional, mas estamos representando diversas variáveis e seus relacionamentos em apenas uma visualização.



Gráficos de Dispersão são amplamente utilizados para representar a relação entre duas variáveis (2 dimensões).

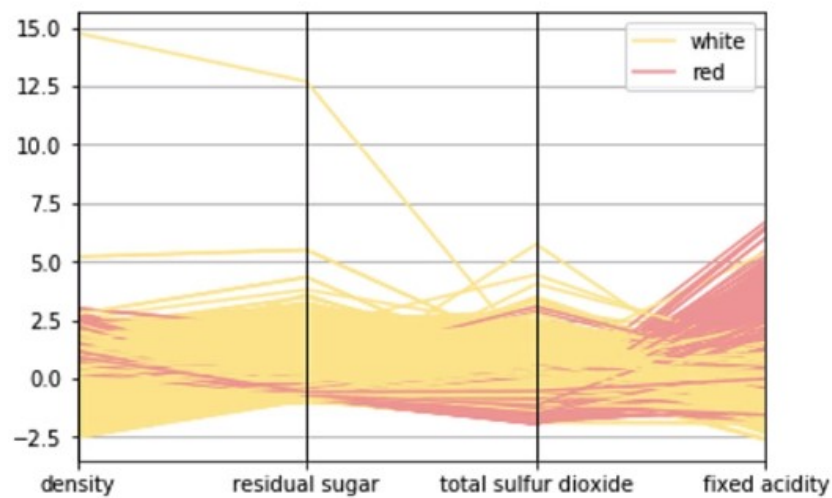


O Pairwise Plot permite visualizar em um mesmo grid diversas relações bidimensionais com gráficos de dispersão. Mas observe que na diagonal temos gráficos de análise univariada através de gráficos de densidade.

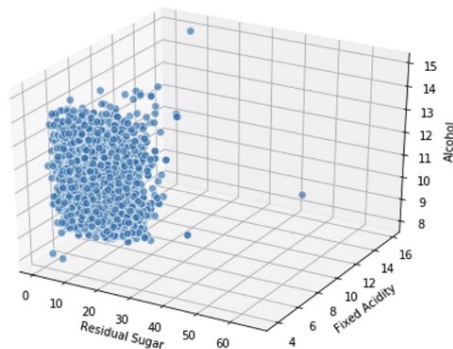


Análise Multivariada – 3 ou + Dimensões:

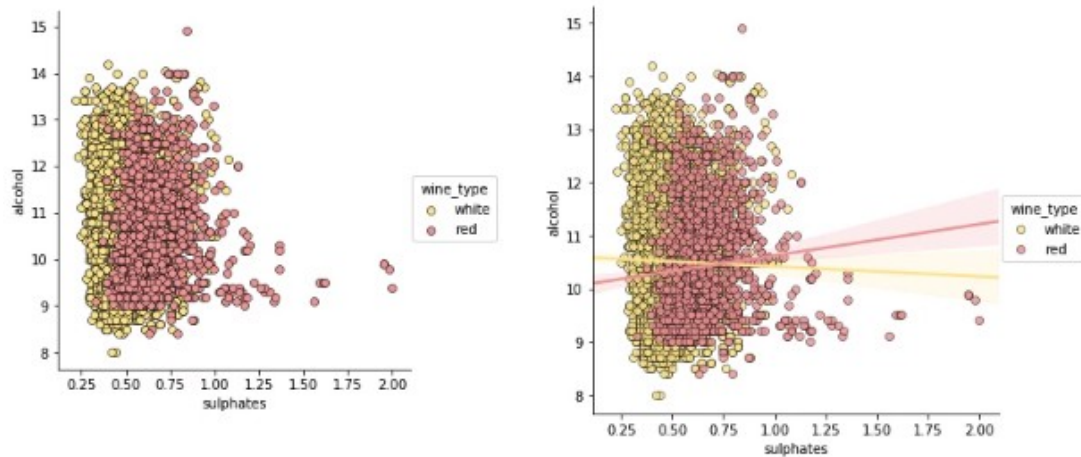
Coordenadas Paralelas são um exemplo de gráfico para análise multivariada com 3 ou mais dimensões.



Gráficos 3D dificilmente serão a visualização ideal em dados multidimensionais.



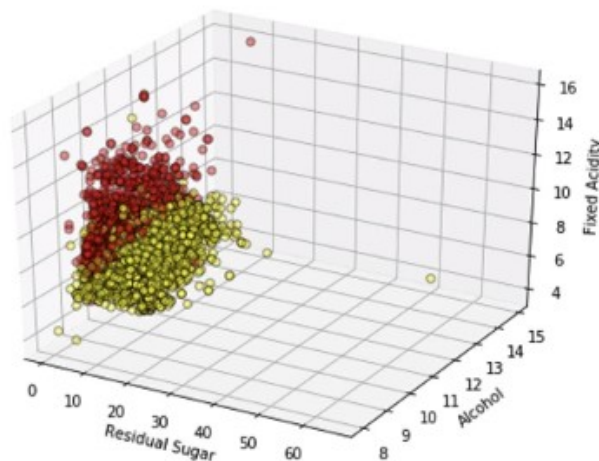
Gráficos de Dispersão também podem ser usados com 3 variáveis (3 dimensões).



As cores representam a terceira dimensão.

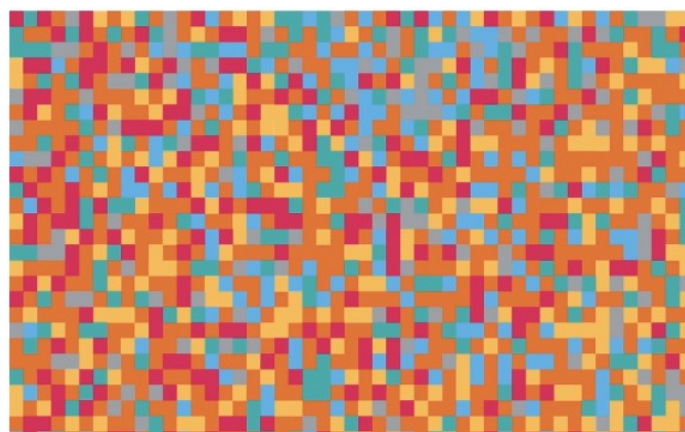
Não use Gráficos 4D, 5D ou 6D. O cérebro humano não consegue processar.

Wine Residual Sugar - Alcohol Content - Acidity - Type



Técnicas Baseadas em Pixels

Em técnicas baseadas em pixels a ideia básica consiste em usar um pixel para representar cada valor de atributo, colorindo-o conforme um mapa de cores previamente fixado de acordo com a faixa de possíveis valores do atributo, sendo que cada um desses atributos tem sua representação visual exibida em sub-janelas individuais na visualização.



Se um único atributo for apresentado em uma janela com resolução 1280 x 1024 é possível exibir mais de um milhão de valores simultaneamente. E essa é uma das grandes vantagens desse tipo de técnica: a grande quantidade de informação que pode ser exibida simultaneamente.

Para o sucesso na aplicação desse tipo de técnica, alguns aspectos precisam receber atenção, como o arranjo dos pixels nas janelas.

Iconografia

As técnicas dessa classe têm como principal característica o uso de ícones como forma de mapear os valores dos atributos de um item de dado multidimensional. Cada característica visual do ícone corresponde a um atributo. E existem diversas formas de fazer isso.

Dependendo da configuração adotada, essa técnica consegue demonstrar grande quantidade de itens de dados, mas há limitações quanto à quantidade de dimensões que podem ser mapeadas sem que haja detrimento na capacidade de representar características detectáveis dos dados.

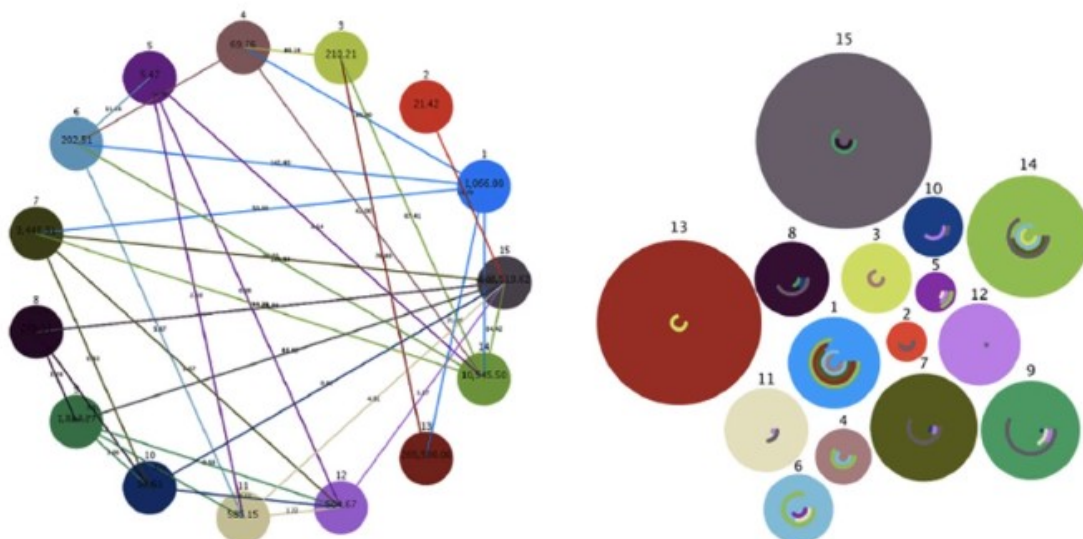
Hierarquia de Visualização

O uso dessa técnica é particularmente interessante na detecção de agrupamentos, de pontos com comportamentos discrepantes e de padrões.

Grafos

Sete pontes de Königsberg é um famoso problema histórico da matemática resolvido por Leonhard Euler, em 1736, cuja solução originou a teoria dos grafos.

A ideia básica dessa categoria é visualizar grafos volumosos usando técnicas que mapeiem as características (direcionado/não-direcionado, cíclico/acíclico, etc.) de um dado grafo, de maneira clara e rápida. As técnicas podem ser subdivididas segundo a dimensionalidade visual da representação: 2D ou 3D.



As visualizações geradas dependem de muitos fatores que refletem características inerentes aos grafos, muitas delas refletindo definições próprias da teoria dos grafos. Esses gráficos são usados para representar agrupamentos e relacionamentos em redes (sociais, logística, transporte, etc.).

Fatores que Influenciam na Leitura e Construção de um Gráfico

1. Escala
2. Ordenação e organização dos valores que são desenhados no gráfico
3. Os papéis da legenda e ícones
4. Como deve estar a posição do gráfico em relação ao texto (“leitura em Z”)
5. O tamanho, a estrutura e a proporção do gráfico
6. A cor pode influenciar na leitura de um gráfico

Escolha de Técnicas de Visualização de Dados

Escala: Se o gráfico tiver poucas barras, opte por um gráfico quadrado.

Principais Tipos de Gráficos por Categoria:

Categorias		Nome da técnica
1D a 3D		Histograma
		Diagrama de caixa
		Gráfico de dispersão (<i>Scatter plot</i>)
		Gráfico de contorno
Multidimensionais	Iconográficas	Faces de Chernoff
		<i>Star glyphs</i>
		Figuras de aresta (<i>Stick figure</i>)
	Geométricas	Matriz de dispersão
		Coordenadas Paralelas
	Orientadas a pixel	<i>Query-dependent techniques</i>
<i>Query-independent techniques</i>		
Hierárquicas ou Baseadas em grafos		Grafos
		<i>Cone trees</i>
		<i>Treemap</i>
		Empilhamento de dimensões (<i>Dimensional stacking</i>)
		Gráfico de mosaico (<i>Mosaic plot</i>)

Categoria	Conceitos				
1D a 3D	Tarefa	Tipo de dado	Dimensão	Volume	Posição
Histograma	Análise da distribuição de frequência, verificação de padrões e detecção de <i>outliers</i>	Quaisquer dados qualitativos e quantitativos	1-D	Pequeno	Não influencia
Diagrama de caixa	Idem Histograma	Quantitativos contínuos	1-D		
Gráfico de dispersão	Verificar correlação entre dois atributos e detecção de <i>outliers</i>	Quantitativos contínuos	2-D		
Gráfico de contorno	Verificar correlação de um atributo em função de outros dois	Quantitativos discretos e contínuos	3-D		

Categoria	Conceitos				
Iconográficas	Tarefa	Tipo de dado	Dimensão	Volume	Posição
Faces de Chernoff	Detecção de agrupamentos e <i>outliers</i> , identificação de padrões de comportamento (Bruckner, 1978)	Quantitativos discretos e contínuos	Até 18 atributos (Chernoff, 1973; Bruckner, 1978)	Pequeno	Não influencia
Star Glyphs	Detecção de agrupamentos e <i>outliers</i>	Quantitativos discretos e contínuos	Até 80 atributos (Rabelo, 2007)	Pequeno	Não influencia
Figuras de Aresta	Detecção de agrupamentos nos dados conforme a composição de diferentes texturas formadas pelos ícones.	Quantitativos discretos e contínuos	De 5 a 15 atributos (Keim, 2005)	Médio	Influencia

Categoria	Conceitos				
Geométricas	Tarefa	Tipo de dado	Dimensão	Volume	Posição
Matriz de dispersão	Correlação entre atributos e detecção de agrupamentos por meio de cores	Quantitativos contínuos	Dezenas de atributos (em torno de 15, segundo Rabelo(2007))	Médio	Não influencia
Coordenadas Paralelas	Visão geral dos dados, identificação de agrupamentos, <i>outliers</i> e correlação entre atributos	Quantitativos e qualitativos	Centenas de atributos (Inselberg, 2008)	Médio	Influencia

Categoria	Conceitos				
Orientadas a pixel	Tarefa	Tipo de dado	Dimensão	Volume	Posição
Dirigido a dados (<i>Query-independent technique</i>)	Detectar padrões e correlação entre os atributos	Dados quantitativos (ex. dados temporais)	10 a 100 atributos (Keim, 2005)	Grande	Influencia
Dirigido a resultados (<i>Query-dependent technique</i>)	Verificar relacionamento dos atributos diante de um resultado de uma consulta (query)	Dados quantitativos (cuos atributos estão no contexto da consulta)			

Categoria	Conceitos				
Grafos	Tarefa	Tipo de dado	Dimensão	Volume	Posição
Grafos	Visão geral da estrutura de relação entre os elementos	Dados com estrutura relacional (hierarquia ou rede)	Alta	Pequeno a médio	Influencia
<i>Cone Trees / Cam Trees</i>	Facilitar a navegação pela estrutura de árvore	Adequado para a visualização de estruturas de arquivos e diretórios	Alta (pode exibir cerca de 1.000 dados)	Pequeno a médio	Não influencia
<i>Treemap</i>	Visualizar agrupamento de dados	Dados que apresentam alguma relação de hierarquia e/ou taxonomia	Alta (pode representar 2 a 20 agrupamentos)	Médio	Influencia
Gráfico de mosaico	Visão geral de atributos qualitativos	Qualitativos nominais e ordinais	Média	Pequeno	Influencia
Empilhamento de dimensões	Deteção de padrões, agrupamentos e outliers	Quantitativos discretos e contínuos	Média	Médio	Influencia

Dashboard Analítico Interativo de Vendas com Dash em Python

Definindo o Problema:

A empresa XPTO Inc comercializa diversos tipos de produtos nos EUA. Os gestores solicitaram um Dashboard analítico e interativo para monitorar as vendas sob diferentes perspectivas.

O Dashboard deve contar com resumos sobre os tipos de entrega dos produtos, regiões onde ocorrem as vendas, segmentos do mercado e categorias dos produtos vendidos. Seria interessante visualizar a margem de lucro em todo território nacional. Como a empresa concede descontos, precisa saber como os descontos afetam a margem de lucro. Outras informações podem ser relevantes com base na análise de dados.

Trabalharemos com dados fictícios e construiremos um Dashboard com Dash, biblioteca para aplicações web em Python.

```
# Dashboard Analítico Interativo de Vendas com Dash em Python

# Execute: pip install -r requirements.txt

# Imports
import dash
import plotly
import locale
import numpy as np
import pandas as pd
import dash_core_components as dcc
import dash_html_components as html
import dash_bootstrap_components as dbc
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from dash.dependencies import Input, Output, State
import warnings
warnings.filterwarnings("ignore")

# Carregando os dados
df = pd.read_csv('dados/dataset.csv')

# Cálculo da margem de lucro bruto
df['Margem_Lucro'] = np.multiply(np.divide(df['Lucro'], df['Venda']), 100).round(2)
```

```

# Função para agrupamento
def group_by(df,col):

    # Agregação
    grouped = df.groupby(by = col, as_index = False).agg({'Venda':'sum',
                                                            'Lucro':'sum',
                                                            'Quantidade':'sum',
                                                            'Desconto':'mean'})

    # Calculando a margem de lucro
    grouped['Margem_Lucro'] = np.multiply(np.divide(grouped['Lucro'], grouped['Venda']), 100).round(2)

    return grouped

# Variáveis para formatação
title_font = {'size':20,'color':'black'}
legend_font = {'size':16,'color':'black'}
global_font = dict(family = "Roboto")

# BoxPlot de desconto vs margem de lucro bruto
figura_1 = px.box(df,
                  x = 'Desconto',
                  y = 'Margem_Lucro',
                  title = 'Comportamento da Margem de Lucro Bruto Por Faixa de Desconto',
                  labels = {'Desconto':'Desconto do Produto',
                            'Margem_Lucro':'Margem de Lucro Bruto'}).update_traces(marker = {'color':'#3399CC'}).update_layout(height = 500,
                                                                                               width = 900,
                                                                                               title = {'font':title_font,
                                                                                               'x':0.5,
                                                                                               'y':0.9,
                                                                                               'xanchor':'center',
                                                                                               'yanchor':'middle'},
                                                                                               font = global_font,
                                                                                               legend = {'font':legend_font},
                                                                                               font_color = 'black',
                                                                                               plot_bgcolor = 'rgba(0,0,0,0)',
                                                                                               paper_bgcolor = 'rgba(0,0,0,0)')

figura_1.add_hline(y = 0,
                  line_dash = "dot",
                  annotation_text = "Lucro Zero",
                  annotation_position = "bottom right")

figura_1.add_vrect(x0 = 0.35,
                  x1 = 0.45,
                  annotation_text = "Declínio",
                  annotation_position = "top left",
                  fillcolor = "red",
                  opacity = 0.20,
                  line_width = 0)

figura_1.add_vline(x = 0.41,
                  line_width = 1,
                  line_dash = "dash",
                  line_color = "red")

# Sunburst Plot
figura_2 = px.sunburst(data_frame = df,
                      path = ['Categoria', 'Sub-Categoria'],
                      values = 'Quantidade',
                      color = 'Lucro',
                      color_continuous_scale = 'rainbow',
                      hover_data = {'Quantidade':True, 'Lucro':True},)

figura_2.update_traces(textfont = {'family':'arial'},
                      textinfo = 'label+percent entry',
                      insidetextorientation = 'radial',
                      marker = {'line':{'color':'black'}})

figura_2.update_layout(title = {'text':'Quantidade Vendida e Lucro Para Cada Tipo de Produto', 'font':title_font, 'x':0.5, 'y':0.02, 'xanchor':'center',
                                'yanchor':'bottom'},
                      legend = {'font':legend_font},
                      font_color = 'black',
                      font = global_font,
                      plot_bgcolor = 'rgba(0,0,0,0)',
                      paper_bgcolor = 'rgba(0,0,0,0)')

# Função para agrupar os dados por estado
def agrupa_estados(dataframe):

```



```

# Dataframe de estados agrupados
estados = dataframe.groupby(['Estado', 'Codigo_Estado', 'Regiao'], as_index = False).agg({'Venda': 'sum',
                                                                                       'Lucro': 'sum',
                                                                                       'Desconto': 'mean',
                                                                                       'Quantidade': 'sum'})

# Calculando a margem de lucro relativo
estados['Margem_Lucro'] = np.multiply(np.divide(estados['Lucro'], estados['Venda']), 100).round(2)

# Ordenação
estados = estados.sort_values('Venda', ascending = False, ignore_index = True)

return estados

# Agrupa os dados por estado
estados_usa = agrupa_estados(df)

# Choropleth Map
us_map = px.choropleth(data_frame = estados_usa,
                       locationmode = 'USA-states',
                       locations = 'Codigo_Estado',
                       scope = 'usa',
                       color = 'Margem_Lucro',
                       color_continuous_scale = 'greens_r',
                       color_continuous_midpoint = 0,
                       hover_name = 'Estado',
                       hover_data = {'Estado': False, 'Venda': True, 'Desconto': True, 'Codigo_Estado': False, 'Regiao': True},
                       labels = {'Margem_Lucro': 'Margem de Lucro Bruto', 'Desconto_mean': 'Desconto Médio'},)

us_map.update_layout(title = {'text': 'Margem de Lucro Bruto - Mapa USA', 'font': {'title_font', 'x': 0.5, 'y': 0.9, 'xanchor': 'center', 'yanchor': 'middle'},
                        font = global_font,
                        font_color = 'black',
                        geo = dict(bgcolor = 'rgba(0,0,0,0)'),
                        paper_bgcolor = 'rgba(0,0,0,0)',
                        plot_bgcolor = 'rgba(0,0,0,0)')

##### App Dash #####

# Criando app dash
app = dash.Dash(__name__,
                external_stylesheets = [dbc.themes.YETI],
                suppress_callback_exceptions = True,
                meta_tags = [{ 'name': 'viewport', 'content': 'width=device-width, initial-scale=1.0' }])

server = app.server

##### Barra Lateral #####

# Componente sidebar
sidebar = html.Div(
    [
        html.H4("Dashboard Analítico", className = "text-white p-1", style = {'marginTop': '1rem'}),
        html.Hr(style = {"borderTop": "1px dotted white"}),
        dbc.Nav(
            [
                dbc.NavLink("Visão Geral", href="/", active="exact"),
                dbc.NavLink("Análise Financeira", href="/pagina-1", active="exact"),
                dbc.NavLink("Conclusão", href="/pagina-2", active="exact"),
            ],
            vertical = True,
            pills = True,
            style = {'fontSize': 16}
        ),
        html.P(u"Versão 1.0", className = 'fixed-bottom text-white p-2'),
    ],
    className = 'bg-dark',

    style = {"position": "fixed",
            "top": 0,
            "left": 0,
            "bottom": 0,
            "width": "14rem",
            "padding": "1rem", },
)

```

Home Page

Tabs Style

```
tab_style = {'border': '1px solid black', 'padding': '6px', 'fontWeight': 'bold', 'margin': '0.5rem',}
tab_selected_style = {'border': '1px solid white', 'background-color': '#3298CC', 'padding': '6px', 'margin': '0.5rem'}
```

Formatar os números decimais

```
locale.setlocale(locale.LC_ALL, "")
```

Container

```
h_container = dbc.Container(
    [
        dbc.Row(
            [
                dbc.Col(
                    dcc.Tabs(id = "radio_options",
                        value = "Transactions",
                        children = [dcc.Tab(label="Total de Transações",
                            value="Transactions",
                            style=tab_style,
                            selected_style = tab_selected_style,),
                        dcc.Tab(label = 'Vendas',
                            value = 'Venda',
                            style = tab_style,
                            selected_style = tab_selected_style,),
                        dcc.Tab(label = 'Lucro',
                            value = 'Lucro',
                            style = tab_style,
                            selected_style = tab_selected_style,),
                        dcc.Tab(label = 'Quantidade',
                            value = 'Quantidade',
                            style = tab_style,
                            selected_style = tab_selected_style,),
                        dcc.Tab(label = 'Desconto Médio',
                            value = 'Desconto',
                            style = tab_style,
                            selected_style = tab_selected_style,)
                    ],
                ),
            ],no_gutters = True, justify = 'around',
        ),
        dbc.Row(
            [
                dbc.Col(
                    [
                        html.P("Total de Vendas",
                            style = {'margin': '1rem', 'textAlign': 'center', 'border': '1px solid white'},
                            className = 'text-white rounded-lg shadow p-1 bg-dark',
                        ),
                        html.P('R$ {}'.format(str(locale.format("%.4f", df.Venda.sum().round(2), grouping=True))),
                            style = {'textAlign': 'center', 'fontColor': 'black'}),
                        html.P('Lucro Total',
                            style = {'margin': '1rem', 'textAlign': 'center', 'border': '1px solid white'},
                            className = 'text-white rounded-lg shadow p-1 bg-dark',
                        ),
                        html.P('R$ {}'.format(str(locale.format("%.4f", df.Lucro.sum().round(2), grouping=True))),
                            style = {'textAlign': 'center', 'color': 'black'}),
                    ],width = 2, style = {"border": "2px solid black", 'borderRight': False},
                ),
                dbc.Col(
                    [
                        dcc.Graph(id = 'subplot',figure = {})
                    ],width = {'size': 5, 'offset': 0}, style = {"border": "2px solid black"},
                ),
                dbc.Col(
                    [
                        dcc.Graph(id = 'map',figure = us_map)
                    ],width = {'size': 5, 'offset': 0}, style = {"border": "2px solid black", 'borderLeft': False})
            ],no_gutters = True, justify = 'around',
        ),
    ],
```

```

    dbc.Row([
        dbc.Col(dcc.Graph(id = 'bar',figure = {}), style={"border": "2px solid black", 'borderTop':False})
    ],no_gutters = True, justify = 'around'),
], fluid = True,
)

##### Layout da Página 1 #####

# Tab style
tab_style = {'border': '1px solid black', 'padding': '6px', 'margin':'1rem', 'fontWeight': 'bold',}
tab_selected_style = {'border':'1px solid white', 'background-color': '#3399CC', 'color':'white', 'margin':'1rem', 'padding': '6px'}

# Container
p1_container = dbc.Container(
    [
        dbc.Row(
            [
                dbc.Col([
                    dcc.Graph(id = 'heat', figure = {}),

                    dcc.Tabs(id = "tabs",
                        value = "Lucro",
                        children = [dcc.Tab(label = "Quantidade", value = "Quantidade", style = tab_style,selected_style = tab_selected_style),
                            dcc.Tab(label = "Lucro", value = "Lucro", style = tab_style, selected_style = tab_selected_style)],
                        vertical = False,)
                ], width = {'size':6},style={"border": "1px solid black",}),

                dbc.Col(
                    dcc.Graph(id = 'sunburst', figure = figura_2, responsive = True),
                    width = {'size':6}, style = {"border": "1px solid black",}),
            ],no_gutters = True, justify = 'around',
        ),
        dbc.Row(
            [
                dcc.Graph(id = 'box', figure = figura_1, responsive = True),

            ], no_gutters = True, justify = 'around',
        ),
    ], fluid = True,
)

##### Layout da Página de Insights #####

# Cria os cards

# Card 1
card_main = dbc.Card(
    [
        dbc.CardHeader(html.H4("Insights", className = "card-title"), className = 'bg-primary text-white'),
        dbc.CardBody(
            [
                dbc.ListGroup(
                    [
                        dbc.ListGroupItem("1) Total de 9960 transações."),
                        dbc.ListGroupItem("2) Média de Vendas de R$ 230.14 por transação."),
                        dbc.ListGroupItem("3) Média de Lucro de R$ 28.69 por transação"),
                    ],
                ),
            ], className = 'bg-info',
        ),
    ],
)

# Card 2
card_con = dbc.Card(
    [
        dbc.CardHeader(html.H4("Problemas Detectados", className = "card-title"), className = 'bg-primary text-white'),
        dbc.CardBody(
            [
                dbc.ListGroup(
                    [
                        dbc.ListGroupItem("1) A margem de lucro bruto parece estar diminuindo com o aumento do desconto nos produtos."),
                        dbc.ListGroupItem("2) Além de 40% de desconto, a loja sofreu apenas perdas."),
                        dbc.ListGroupItem("3) As quantidades vendidas não estão aumentando com o desconto maior."),
                    ],
                ),
            ],
        ),
    ],
)

```

```

        ],
    ),
    ],className='bg-info',
),
],
)

# Card 3
card_final = dbc.Card(
    [
        dbc.CardHeader(html.H4("Conclusão", className = "card-title"), className = 'bg-primary text-white'),
        dbc.CardBody(
            [
                html.P(["A loja tem um bom desempenho quando nenhum desconto ou desconto inferior a 20% é aplicado.", html.Hr(),
                    "A loja pode se beneficiar reduzindo o desconto em itens de produtos deficitários.", html.Hr(),
                    "O marketing e a propaganda em regiões com menor base de clientes podem ajudar a aumentar a presença da marca."],
                    ],
                className = "card-text",
            ),
            ],className = 'bg-info',
        ),

        dbc.CardLink("Suporte", className = 'text-center font-weight-bold', href = "https://www.datascienceacademy.com.br")
    ],
    color = 'primary',
    outline = True,
)

# Container
p2_container = dbc.Container(
    [
        dbc.Row(
            [
                dbc.Col(card_main, width = {'size':4,'offset':1}, style={'marginTop':'2rem'}),
                dbc.Col(card_con, width = {'size':4,'offset':1}, style={'marginTop':'2rem'}),
            ], no_gutters = True, justify = "around"
        ),

        dbc.Row(
            [
                dbc.Col(card_final, width = {'size':6}, style={'marginTop':'2rem', 'marginBottom':'2rem'}),
            ], no_gutters=True,justify="around"
        ),
    ]
)

##### Layout Principal #####

CONTENT_STYLE = {"marginLeft": "13rem",
    "margin-right": "1rem",
    "padding": "0rem 0rem",
    'background-color':'#F0F4F5',}

content = html.Div(id = "page-content", children = [], style = CONTENT_STYLE )

##### Layout Geral #####

app.layout = html.Div(
    [
        dcc.Location(id = "url"),
        sidebar,
        content
    ]
)

### App Callback Functions ###

# Calback para renderização das páginas
@app.callback(Output("page-content", "children"), [Input("url", "pathname")])

def render_page_content(pathname):
    if pathname == "/":
        return [h_container]
    elif pathname == "/pagina-1":
        return [p1_container,]

```

[illegible]

```

        legend = {'font':legend_font},
        font_color = 'black',
        paper_bgcolor = 'rgba(0,0,0,0)',
        plot_bgcolor = 'rgba(0,0,0,0)')

    return fig, figura_3

# Callback do mapa de calor (gráfico de pixels)
@app.callback(Output(component_id = 'heat', component_property = 'figure'), [Input(component_id = 'tabs', component_property = 'value')])

def update_output(tab):
    dff = df.copy()
    dff['Margem_Lucro'] = np.multiply(np.divide(dff['Lucro'],dff['Venda']),100).round(2)
    dff.round(2)

    pro = pd.crosstab(index = dff['Desconto'], columns = dff['Sub-Categoria'], values = dff[tab], aggfunc = np.sum )

    figura_3 = px.imshow(pro,
        color_continuous_scale = 'greens_r',
        title = '{} em Toda a Gama de Descontos em Produtos'.format(tab),
        labels = {'color':tab}
    ).update_layout(title = {'font':title_font,
        'x':0.5, 'y':0.9,
        'xanchor':'center', 'yanchor':'middle'},
        font = global_font,
        legend = {'font':legend_font},
        font_color = 'black',
        plot_bgcolor = 'rgba(0,0,0,0)',
        paper_bgcolor = 'rgba(0,0,0,0)')

    return figura_3

# Executa a app
if __name__ == '__main__':
    app.run_server(debug = False, use_reloader = False)

```

<https://plotly.com/dash/>

Para instalação dos pacotes necessários, execute o comando abaixo (dentro da pasta da aplicação):
pip install -r requirements.txt

Quizz

UI Design, ou User Interface Design (Design de Interface do Usuário), é o meio pela qual uma pessoa interage e controla um dispositivo, software ou aplicativo.

O conjunto de elementos visuais que comunicam e representam os valores, posicionamento e diferenciais competitivos de uma empresa ou organização é chamado de Identidade Visual.

A integração de técnicas de Big Data Analytics e Visualização é referenciada na literatura como Visual Data Mining.

A construção de gráficos e os cuidados que deve-se ter ao fazê-lo, de modo que o gráfico possa de fato alcançar o objetivo de transmitir informações ao leitor, deve garantir que a informação seja transmitida de maneira clara e correta.

A Web Semântica incorpora significado às informações da web. Isso proporciona um ambiente onde máquinas e usuários trabalhem em conjunto. Tendo cada tipo de informação devidamente identificada, fica fácil para os sistemas encontrarem informações mais precisas sobre um determinado assunto.

9.4. Dashboard Design

O Que São Dashboards?

Um Dashboard é uma exibição visual das informações mais importantes necessárias para alcançar um ou mais objetivos, consolidado e organizado em uma única tela para que as informações possam ser monitoradas ao mesmo tempo.

Um dashboard é uma ferramenta de gestão e visualização de informações que é usado para monitorar KPI's, métricas e outros pontos de dados relevantes para o negócio, departamento ou projeto. Com o uso de visualizações de dados, o dashboard simplifica o complexo processo de análise de dados e provê ao usuário uma visão clara da situação atual ou eventuais previsões.

Os dashboards receberam esse nome a partir dos painéis de automóveis.

Principais Características dos Dashboards

- Ele se encaixa em uma tela, mas pode haver barras de rolagem para tabelas com demasiadas linhas ou gráficos com muitos pontos de dados.
- É altamente interativo e geralmente fornece funcionalidades como filtragem e drill-downs (navegar por uma hierarquia).
- É usado principalmente para encontrar correlações, tendências, outliers (anomalias), padrões e condições de negócios em dados.
- Os dados usados em uma ferramenta de análise visual são geralmente dados históricos. No entanto, é possível construir dashboards para visualização de dados em tempo real.
- Ele ajuda a identificar indicadores de desempenho.
- É tipicamente utilizado por usuários tecnicamente experientes como analistas de dados e pesquisadores, embora venha sendo cada vez mais utilizado por profissionais de diversas áreas de negócio.

Resumo das principais características:

- É essencialmente um instrumento de apoio à decisão.
- Expõe rapidamente os principais indicadores de uma organização, área, unidade funcional, projeto, etc.
- Tenta apresentar informação em uma única tela.
- Possui uma apresentação gráfica simples, objetiva, mas também elegante.
- Utiliza técnicas de design de modo a reduzir o lixo visual e para dar maior eficácia na transmissão da informação/mensagem.
- Tenta combinar diferentes variáveis sobre diferentes perspectivas de modo a expor relações que seriam difíceis de identificar analisando os mesmos dados em separado.

O dashboard é um tipo especial de relatório que irá auxiliar os responsáveis de uma organização, de uma área, de uma unidade ou simplesmente de um projeto na tomada de decisões de gestão.

Dashboards Operacionais X Dashboards Analíticos X Dashboards Executivos

Dashboard Operacional é utilizado diretamente pelas equipes de trabalho, com foco em determinados processos, possibilitando análises específicas. Esses dados servem, por exemplo, para identificar gargalos e etapas críticas da operação, auxiliando na correção de problemas pontuais e tendências negativas. Com simplicidade, facilitam a comunicação e ainda permitem a interação e atualização de todos os profissionais envolvidos.

Dashboard Analítico é preparado para fornecer informações mais detalhadas e é usado para traçar tendências em relação aos objetivos corporativos pré-determinados, ou seja, para avaliar se os processos e projetos estão evoluindo de acordo com as expectativas, ou ainda, para apresentar o resultado de um projeto de análise preditiva. Com dados constantemente atualizados, é possível perceber rapidamente os resultados de ações internas e também as reações do mercado, em relação aos produtos e serviços lançados. É uma maneira de mensurar os impactos causados por cada decisão tomada, sendo possível, assim, interferir e corrigir os desvios com maior rapidez, evitando desperdícios e prejuízos. Exemplo: google analytics.

Dashboard Executivo permite a visualização de grandes quantidades de informações, devidamente estratificadas, formatadas, consolidadas e organizadas, de modo a facilitar o processo de tomada de decisões. Utiliza recursos visuais e gráficos para facilitar a compreensão geral. É muito útil para monitorar indicadores chave de desempenho, possibilitando uma análise completa sobre todos os processos de uma empresa. É o chamado dashboard de negócio e utilizado, principalmente, para facilitar e agilizar a tomada de decisões.

Principais Tipos de Dashboards

- **Dashboard Financeiro:** demonstrativo de resultados, fluxo de caixa, contas a pagar e contas a receber, variação, total do mês, acumulado do ano, etc. Pode ser tanto operacional, como executivo.
- **Dashboard Comercial:** desempenho de equipe de vendas, produtos, fornecedores, descontos, preço médio, ticket médio, volume de vendas, faturamento, etc.
- **Dashboard de Marketing:** geração de leads, desempenho por campanha de marketing, canal que gere tráfego ou novos clientes, relação entre perfil do cliente e número de vendas, comparativos por setor, segmento, região, evolução mensal, etc. Normalmente, trata-se de um dashboard analítico.
- **Dashboard de RH:** quadro de pessoal, força de trabalho, rotatividade, perfil da área de negócio, indicadores de desempenho, turnover (demissões e contratações), percentual de ocupação, custo de horas extras, análise real x análise de metas dos colaboradores, etc. Podem ser dashboards analíticos ou operacionais.
- **Dashboard de Logística:** usado para permitir uma visão gerencial da cadeia de suprimentos, por meio de indicadores de desempenho. Pedidos em atraso, taxa de evolução de pedidos, desempenho por centro de distribuição, categoria, desempenho por canal de distribuição, percentual de devolução de produtos, pedidos atrasados, fretes, quantidades e valores em estoque, etc. Tipicamente é um dashboard operacional.

- Dashboard de Gerenciamento de Projetos: estágios do projeto, cronograma, atividades previstas, interdependências, percentual de conclusão do projeto, etc.
- Dashboard de Atendimento ao Cliente (Call Center): número de chamadas por período, duração de cada atendimento, casos resolvidos ou aguardando outras providências, natureza do contato, avaliação de satisfação do cliente, etc. Tipicamente é um dashboard operacional.

Passo a Passo do Processo de Design de Dashboards

Projetado corretamente, um dashboard aumenta a produtividade para todos os usuários.

São 6 etapas do processo de design de dashboards:

1. Conheça seu público
2. Projeto (wireframe)
3. Conheça sua plataforma
4. Agora sim: Design
5. Não esqueça a UX
6. Obtenha feedback

Não existe nada mais sofisticado que a simplicidade.

Desenvolvimento do Layout

Organização:

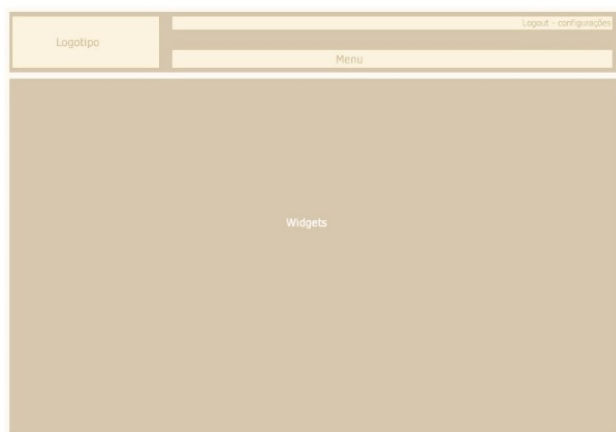
Grid: pequenos blocos formando linhas e colunas. Ajuda a pensar de forma modular, com pequenos blocos de conteúdo a serem posicionados no dashboard. A recomendação geral é utilizar entre 10 a 12 blocos. Blocos maiores devem ser usados para informações mais relevantes.

Cores: procure utilizar cores leves, suaves e evite cores muito quentes, fortes (muito brilhantes) e escuras. Utilize cores que já se estejam acostumados no dia a dia.

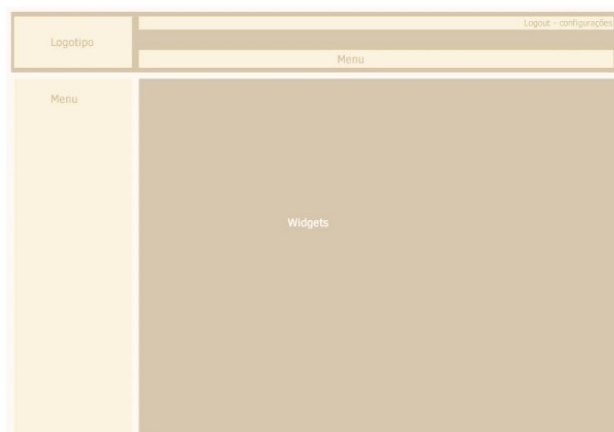
Áreas de Atenção: área de atenção é a diagonal da leitura em “Z”.

Formatos de Layout:

Exemplo 1



Exemplo 2



O Que Deve Fazer Parte de um Dashboard?

- Métricas e Indicadores de Desempenho
- Envio de E-mails
- Possibilidade de Detalhamento das Informações
- Recuperação Rápida de Informação
- Taxa de Atualização das Informações no Dashboard
- Recomendações e/ou Previsões
- Versionamento
- Filtros e Drill-Down/Up

Dashboards – Desenvolver ou Usar Soluções Proprietárias

Desenvolver	Usar Soluções Proprietárias
Habilidades em Linguagem de Programação Python, R ou JavaScript	Habilidades em Power BI, Tableau, QlikSense
Maior flexibilidade	Menor flexibilidade
Não há pagamento de licença de software	Há pagamento de licença de software, em alguns casos por usuário
Mais complexo e requer mais tempo de construção do Dashboard	Menos complexo e requer menos tempo de construção do Dashboard

O Poder da Narrativa Visual

Se a história fosse ensinada na forma de histórias, nunca seria esquecida!

Executivos e gerentes estão sendo bombardeados com dashboards cheios de análise. E muitas vezes isso não é suficiente para tomada de decisão baseada em dados, porque eles não sabem a história por trás dos dados.

Quando dados e histórias são usados em conjunto, eles ressoam com o público em um nível intelectual e emocional.

Narratologia Visual é a ciência que estuda a Narrativa Visual.

As características da narrativa visual incluem:

- Uma história persuasiva com um ponto de vista.
- Imagens de alta qualidade, imóveis ou em movimento.
- Valor social, ambiental ou espiritual.
- Um apelo (explícito ou implícito) para a transformação de atitudes e comportamentos.

Recomendações Gerais Para o Design de Dashboards

O que NÃO fazer na criação de dashboards?

- Usar muitas cores
- Inserir muitos efeitos
- Uso demasiado de imagens e ícones
- Gráficos inadequados
- Apresentar dados irrelevantes

O que fazer na criação de dashboards?

- Elabore o dashboard em uma única página
- Realce as informações relevantes
- Construa os elementos de forma consolidada e ofereça a opção de detalhamento
- Utilize poucas cores e suaves
- Configure somente os efeitos e imagens realmente necessários
- Tente incluir uma narrativa visual com uma sequência lógica de gráficos e imagens

Alinhando a Terminologia: Dataviz, BI, Reporting, Analytics e KPI's

Dataviz; abreviação para Data Visualization.

BI (Business Intelligence): é o conceito por trás da análise de dados do passado a fim de gerar relatórios gerenciais e compreender os componentes chaves de uma empresa. É olhar para o passado e compreender o que está acontecendo, gerando relatórios que ajudam na tomada de decisões.

Reporting: processo de geração de relatórios.

Analytics: termo usado para representar o uso de técnicas de ciência de dados.

KPI's (Key Performance Indicators): indicadores chave de performance.

Mini-Projeto 1 – Dashboard Interativo com Streamlit, Folium e Plotly para Monitoramento de Casos de Covid-19 em Tempo Real

Vamos trabalhar agora no Mini-Projeto 1 e construir um Real-Time Dashboard Interativo. Usaremos Linguagem Python e as bibliotecas Streamlit (para construção do Dashboard), Folium (para construção de mapas) e Plotly (para gráficos interativos). Usaremos uma API para acessar dados em tempo real sobre o Covid-19.

Durante o Mini-Projeto 1 vamos praticar muitos dos conceitos estudados neste capítulo e nos capítulos anteriores.

Para reproduzir o projeto você precisa ter o Anaconda Python instalado no seu computador. Visite o Capítulo 1 do Curso Gratuito de Python Fundamentos aqui na DSA para aprender como instalar o Anaconda.

```
# Mini-Projeto 1 - Dashboard Interativo com Streamlit, Folium e Plotly Para Monitoramento de Casos de Covid-19 em Tempo Real

# Execute no terminal: streamlit run Mini-Projeto1.py

# Imports
```

```

import json
import folium
import requests
import mimetypes
import http.client
import pandas as pd
import streamlit as st
import plotly
import plotly.express as px
from streamlit_folium import folium_static
from folium.plugins import HeatMap
from pandas.io.json import json_normalize
import warnings
warnings.filterwarnings("ignore", category = FutureWarning)

# Função Main
def main():

    # Título da área principal
    st.markdown("<h1 style='text-align: center; color: #fa634d;'><strong><u>Real-Time Covid-19 Dashboard</u></strong></h1>",
    unsafe_allow_html = True)

    # Título do menu lateral
    st.sidebar.markdown("<h1 style='text-align: center; color: #baccee;'><strong><u>Monitoramento de Casos de Covid-19</u></strong></h1>",
    unsafe_allow_html = True)

    # Sub-títulos da área principal
    st.markdown("O Dashboard Utiliza Dados Reais da Johns Hopkins CSSE.", unsafe_allow_html = True)
    st.markdown("Os Dados São Atualizados Diariamente.", unsafe_allow_html = True)

    # Conexão aos dados em tempo real via API
    # https://coronavirus.jhu.edu/map.html
    # https://covid19api.com/
    conn = http.client.HTTPSConnection("api.covid19api.com")
    payload = ""
    headers = {}
    conn.request("GET", "/summary", payload, headers)
    res = conn.getresponse()
    data = res.read().decode('UTF-8')
    covid = json.loads(data)

    # Gera o dataframe
    df = pd.DataFrame(covid['Countries'])

    # Remove as colunas que não usaremos
    covid1 = df.drop(columns = ['CountryCode', 'Slug', 'Date', 'Premium'], axis = 1)

    # Realiza os cálculos
    covid1['ActiveCases'] = covid1['TotalConfirmed'] - covid1['TotalRecovered']
    covid1['ActiveCases'] = covid1['ActiveCases'] - covid1['TotalDeaths']

    # Dataframe com agrupamentos
    dfn = covid1.drop(['NewConfirmed', 'NewDeaths', 'NewRecovered'], axis = 1)
    dfn = dfn.groupby('Country')['TotalConfirmed', 'TotalDeaths', 'TotalRecovered', 'ActiveCases'].sum().sort_values(by = 'TotalConfirmed', ascending
= False)
    dfn.style.background_gradient(cmap = 'Oranges')
    dfc = covid1.groupby('Country')['TotalConfirmed', 'TotalDeaths', 'TotalRecovered', 'ActiveCases'].max().sort_values(by = 'TotalConfirmed',
ascending = False).reset_index()

    # Mapa 1
    m1 = folium.Map(tiles = 'Stamen Terrain', min_zoom = 1.5)
    url = 'https://raw.githubusercontent.com/python-visualization/folium/master/examples/data'
    country_shapes = f'{url}/world-countries.json'
    folium.Choropleth(geo_data = country_shapes,
        min_zoom = 2,
        name = 'Covid-19',
        data = covid1,
        columns = ['Country', 'TotalConfirmed'],
        key_on = 'feature.properties.name',
        fill_color = 'YlOrRd',
        nan_fill_color = 'white',
        legend_name = 'Total de Casos Confirmados').add_to(m1)

    # Mapa 2
    m2 = folium.Map(tiles = 'Stamen Terrain', min_zoom = 1.5)
    url='https://raw.githubusercontent.com/python-visualization/folium/master/examples/data'
    country_shapes = f'{url}/world-countries.json'
    folium.Choropleth(geo_data = country_shapes,

```

```

min_zoom = 2,
name = 'Covid-19',
data = covid1,
columns = ['Country', 'TotalRecovered'],
key_on = 'feature.properties.name',
fill_color = 'PuBu',
nan_fill_color = 'white',
legend_name = 'Total de Casos Recuperados',).add_to(m2)

# Mapa 3
m3 = folium.Map(tiles = 'Stamen Terrain', min_zoom = 1.5)
url = 'https://raw.githubusercontent.com/python-visualization/folium/master/examples/data'
country_shapes = f'{url}/world-countries.json'
folium.Choropleth(geo_data = country_shapes,
min_zoom = 2,
name = 'Covid-19',
data = covid1,
columns = ['Country', 'ActiveCases'],
key_on = 'feature.properties.name',
fill_color = 'YlGnBu',
nan_fill_color = 'white',
legend_name = 'Total de Casos Ativos',).add_to(m3)

# Coordenadas dos países
coordinates = pd.read_csv('dados/country-coordinates-world.csv')
covid_final = pd.merge(covid1, coordinates, on = 'Country')

# Mostra o dataframe agrupado na tela
dfn

# Obtém os totais consolidados
confirmed_tot = int(df['TotalConfirmed'].sum())
deaths_tot = int(df['TotalDeaths'].sum())
recovered_tot = int(df['TotalRecovered'].sum())
active_tot = int(df['ActiveCases'].sum())

# Imprime os totais na área principal
st.write("Total de Casos Confirmados no Mundo - ", confirmed_tot)
st.write("Total de Mortes no Mundo - ", deaths_tot)
st.write("Total de Pessoas Recuperadas no Mundo - ", recovered_tot)
st.write("Total de Casos Ativos no Mundo - ", active_tot)

# Menu lateral - Mapas
st.sidebar.subheader('Análise Através de Mapa')

# Caixa de seleção
select = st.sidebar.selectbox('Escolha o Tipo de Mapa', ['Casos Confirmados', 'Casos Recuperados', 'Casos Ativos', 'Mortes'], key = '1')

# Condicional para mostrar o mapa
if not st.sidebar.checkbox("Ocultar Mapa", False):

    if select == "Casos Confirmados":
        folium_static(m1)

    elif select == "Casos Recuperados":
        folium_static(m2)

    elif select == "Casos Ativos":
        folium_static(m3)

    else:
        m4 = folium.Map(tiles = 'StamenToner', min_zoom = 1.5)
        deaths = covid_final['TotalDeaths'].astype(float)
        lat = covid_final['latitude'].astype(float)
        long = covid_final['longitude'].astype(float)
        m4.add_child(HeatMap(zip(lat, long, deaths), radius = 0))
        folium_static(m4)

# Menu lateral - Gráfico de Barras
st.sidebar.subheader('Análise com Gráfico de Barras')

select = st.sidebar.selectbox('Escolha o Gráfico de Barras',
['Casos Confirmados', 'Casos Recuperados', 'Casos Ativos', 'Mortes'],
key = '2')

if not st.sidebar.checkbox("Ocultar Gráfico de Barras", False):

    if select == "Casos Confirmados":

```

```

fig = px.bar(dfc.head(10), y = 'TotalConfirmed', x = 'Country', color = 'Country', height = 400)
fig.update_layout(title = 'Comparação do Total de Casos Confirmados dos 10 Países Mais Afetados Neste Momento',
xaxis_title = 'Country',
yaxis_title = 'Total de Casos Confirmados',
template = "ggplot2")
st.plotly_chart(fig)

elif select == "Casos Recuperados":

fig = px.bar(dfc.head(10), y = 'TotalRecovered', x = 'Country', color = 'Country', height=400)
fig.update_layout(title='Comparação do Total de Casos Recuperados dos 10 Países Mais Afetados Neste Momento',
xaxis_title = 'Country',
yaxis_title = 'Total de Casos Recuperados',
template = "seaborn")
st.plotly_chart(fig)

elif select == "Casos Ativos":

fig = px.bar(dfc.head(10), y = 'ActiveCases', x = 'Country', color = 'Country', height = 400)
fig.update_layout(title = 'Comparação de Casos Ativos dos 10 Países Mais Afetados Neste Momento',
xaxis_title = 'Country',
yaxis_title = 'Total de Casos Ativos',
template = "plotly")
st.plotly_chart(fig)

else:

fig = px.bar(dfc.head(10), y = 'TotalDeaths', x = 'Country', color = 'Country', height = 400)
fig.update_layout(title='Comparação do Total de Mortes dos 10 Países Mais Afetados Neste Momento',
xaxis_title = 'Country',
yaxis_title = 'Total de Mortes',
template = "plotly_dark")
st.plotly_chart(fig)

# Menu lateral - Gráfico 3D
st.sidebar.subheader('Análise com Gráfico 3D')

if not st.sidebar.checkbox("Ocultar Gráfico 3D", True):

fig = px.scatter_3d(dfc.head(10), x = 'TotalConfirmed', y = 'TotalDeaths', z = 'TotalRecovered', color = 'Country')
fig.update_layout(title = 'Gráfico 3D do Total de Casos, Total de Mortes e Total de Recuperados dos 20 Principais Países Afetados Neste
Momento')
st.plotly_chart(fig)

if st.sidebar.checkbox("Mostrar Dados Brutos", False):
st.subheader("Dados do Covid 19")
st.write(covid1)

if __name__ == '__main__':
main()

st.markdown("Obrigado Por Usar Este Real-Time Dashboard!")

```

Quizz

O Dashboard é construído para que os gestores possam ter acesso de forma sistemática à informação mais relevante sobre a performance organizacional da sua instituição, ou seja, a história da sua atividade.

Características de um Dashboard:

- Possui uma apresentação gráfica simples, objetiva, mas também elegante.
- Utiliza técnicas de design de modo a reduzir o “lixo visual” e para dar maior eficácia na transmissão da informação/mensagem.
- É essencialmente um instrumento de apoio à decisão.
- Tenta apresentar informação em apenas uma tela.

Um Dashboard é uma exibição visual das informações mais importantes necessárias para alcançar um ou mais objetivos; Consolidado e organizado em uma única tela para que as informações possam ser monitoradas ao mesmo tempo.

Um Dashboard bem projetado é uma notável ferramenta de gerenciamento de informações.

Um Dashboard é usado principalmente para encontrar correlações, tendências, outliers (anomalias), padrões e condições de negócios em dados.

9.5. Visualização de Dados e Dashboards com Python

Mini-Projeto 2 – Data App – Dashboard Financeiro Interativo e em Tempo Real Para Previsão de Ativos Financeiros

Vamos trabalhar agora no Mini-Projeto 2 e construir uma Data App, ou seja, uma aplicação de dados que envolve gráficos interativos em um Dashboard e previsões com modelo de Machine Learning.

Nossa Data App deve apresentar ao usuário os dados de cotações de ações de 4 empresas: Apple, Petrobras, Uber e Pfizer.

Além da tabela de dados, nossa Data App deve mostrar em um gráfico as cotações de abertura e fechamento das ações de 2015 aos dias atuais. Os dados de cotações de ações podem ser extraídos em tempo real do portal Yahoo Finance.

Uma vez coletados os dados, nossa Data App deve ser capaz de apresentar as previsões de cotações para um horizonte de previsão de 1 a 4 anos (o usuário deve poder selecionar o horizonte de previsão).

A Data App deve mostrar os dados previstos em formato de tabela, bem como no formato de gráfico interativo.

Vamos construir a Data App com Linguagem Python, usando o Streamlit como biblioteca para criação da app, Prophet para criação do modelo de Machine Learning e previsão com séries temporais e os gráficos interativos com o Plotly.

```
# Mini-Projeto 2 - Data App - Dashboard Financeiro Interativo e em Tempo Real Para Previsão de Ativos Financeiros

# Imports
import numpy as np
import yfinance as yf
import streamlit as st
import matplotlib.pyplot as plt
from fbprophet import Prophet
from fbprophet.plot import plot_plotly
from plotly import graph_objs as go
from datetime import date
import warnings
warnings.filterwarnings("ignore")

# Define a data de início para coleta de dados
INICIO = "2015-01-01"

# Define a data de fim para coleta de dados (data de hoje, execução do script)
HOJE = date.today().strftime("%Y-%m-%d")

# Define o título do Dashboard
st.title("Mini-Projeto 2 - Data App")
st.title("Dashboard Financeiro Interativo e em Tempo Real Para Previsão de Ativos Financeiros")

# Define o código das empresas para coleta dos dados de ativos financeiros
# https://finance.yahoo.com/most-active
empresas = ('PBR', 'GOOG', 'UBER', 'PFE')

# Define de qual empresa usaremos os dados por vez
empresa_selecionada = st.selectbox('Selecione a Empresa Para as Previsões de Ativos Financeiros:', empresas)

# Função para extrair e carregar os dados
@st.cache
def carrega_dados(ticker):
    dados = yf.download(ticker, INICIO, HOJE)
    dados.reset_index(inplace = True)
    return dados
```



```

# Mensagem de carga dos dados
mensagem = st.text('Carregando os dados...')

# Carrega os dados
dados = carrega_dados(empresa_selecionada)

# Mensagem de encerramento da carga dos dados
mensagem.text('Carregando os dados...Concluído!')

# Sub-título
st.subheader('Visualização dos Dados Brutos')
st.write(dados.tail())

# Função para o plot dos dados brutos
def plot_dados_brutos():
    fig = go.Figure()
    fig.add_trace(go.Scatter(x = dados['Date'], y = dados['Open'], name = "stock_open"))
    fig.add_trace(go.Scatter(x = dados['Date'], y = dados['Close'], name = "stock_close"))
    fig.layout.update(title_text = 'Preço de Abertura e Fechamento das Ações', xaxis_rangelslider_visible = True)
    st.plotly_chart(fig)

# Executa a função
plot_dados_brutos()

st.subheader('Previsões com Machine Learning')

# Prepara os dados para as previsões com o pacote Prophet
df_treino = dados[['Date','Close']]
df_treino = df_treino.rename(columns = {"Date": "ds", "Close": "y"})

# Cria o modelo
modelo = Prophet()

# Treina o modelo
modelo.fit(df_treino)

# Define o horizonte de previsão
num_anos = st.slider('Horizonte de Previsão (em anos):', 1, 4)

# Calcula o período em dias
periodo = num_anos * 365

# Prepara as datas futuras para as previsões
futuro = modelo.make_future_dataframe(periods = periodo)

# Faz as previsões
forecast = modelo.predict(futuro)

# Sub-título
st.subheader('Dados Previstos')

# Dados previstos
st.write(forecast.tail())

# Título
st.subheader('Previsão de Preço dos Ativos Financeiros Para o Período Seleccionado')

# Plot
grafico2 = plot_plotly(modelo, forecast)
st.plotly_chart(grafico2)

# Fim

```

Mini-Projeto 3 – Data App – Dashboard Interativo Para Detecção de Fraudes com H2O Wave

Vamos trabalhar agora no Mini-Projeto 3 e construir uma Data App com um Dashboard Interativo Para Detecção de Fraudes com H2O Wave.

O H2O Wave é uma ferramenta do framework H2O para construção de gráficos e dashboards interativos em tempo real. A programação é em Python e a ferramenta é super amigável permitindo a criação de dashboards poderosos com poucas linhas de código.

Teremos que preparar nosso ambiente de trabalho, instalar a ferramenta e então construir o dashboard, o que faremos em tempo real usando dados reais disponíveis publicamente. Os arquivos do mini-projeto podem ser encontrados ao final do capítulo.

Depois de conhecer o H2O Wave você provavelmente vai considera-lo em seus próximos projetos de Visualização de Dados.

H2O Wave é uma pilha de software para construir aplicativos e dashboards interativos, de baixa latência, em tempo real e baseados em navegador inteiramente em Python, sem usar HTML, Javascript ou CSS.

Ele se destaca na captura de dados, visualizações e gráficos de várias fontes e na transmissão ao vivo pela web.

H2O Wave oferece a seus programas Python a capacidade de enviar conteúdo para clientes conectados conforme isso acontece, em tempo real. Em outras palavras, ele permite que seu programa exiba informações atualizadas sem solicitar que seus usuários cliquem no botão de recarregar do navegador.

Você pode usar H2O Wave para:

- Dashboards e visualizações para monitoramento ao vivo.
- Exibições de informações ao vivo: notícias, tickers, saúde ou dados de desempenho.
- Aplicativos que requerem notificações instantâneas, atualizações, eventos ou alertas.
- Aplicativos que envolvem mensagens: chat, bots, etc.
- Aplicativos colaborativos: quadros brancos, compartilhamento, etc.
- Você também pode usar H2O Wave quando estiver procurando por um framework de aplicativo web.

[illegible]

```
#####

# Gráfico de Total de Rendimentos Por Tipo de Rendimento e Gênero (Gráfico de Barras)
# https://wave.h2o.ai/docs/layout

# Prepara os dados
df_bar = dataset.loc[:200,['NAME_INCOME_TYPE', 'AMT_INCOME_TOTAL', 'CODE_GENDER']]

# Gráfico
grafico1 = page.add('bar_plot', ui.plot_card(box = '1 3 4 4',
                                             title = 'Total de Rendimentos Por Tipo de
                                             Rendimento e Gênero',
                                             data = data(fields = df_bar.columns.tolist()),
                                             rows = df_bar.values.tolist(),
                                             plot = ui.plot(marks = [ui.mark(type =
'interval',
x = '=NAME_INCOME_TYPE',
y = '=AMT_INCOME_TOTAL',
color = '=CODE_GENDER',
dodge = 'auto'))]))

#####

# Gráfico de Clientes Por Escolaridade (Gráfico de Dispersão)

# Prepara os dados
df_point = dataset.loc[:200,['DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'NAME_EDUCATION_TYPE']]

# Gráfico
grafico2 = page.add('point_plot', ui.plot_card(box = '5 3 5 2',
                                             title = 'Clientes Por
                                             Escolaridade',
                                             data = data(fields =
df_point.columns.tolist(), rows = df_point.values.tolist()),
                                             plot = ui.plot([ui.mark(type =
'point',
x = '=DAYS_REGISTRATION',
y = '=DAYS_ID_PUBLISH',
color = '=NAME_EDUCATION_TYPE'))]))

#####

# Gráfico de Total de Rendimentos Por Total de Crédito (Gráfico de Bolhas)

# Prepara os dados
df_point_sized = dataset.loc[:200,['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY']]

# Gráfico
grafico3 = page.add('point_plot_sized', ui.plot_card(box = '5 5 5 2',
                                             title = 'Total de
                                             Rendimentos Por Total de Crédito',
                                             data = data(fields =
df_point_sized.columns.tolist(), rows = df_point_sized.values.tolist()),
                                             plot =
ui.plot([ui.mark(type = 'point',
x = '=AMT_INCOME_TOTAL',
y = '=AMT_CREDIT',
size = '=AMT_ANNUITY'))]))

#####

# Gráfico de Total de Rendimentos Por Tipo de Rendimento e Status Familiar (Gráfico de Barras Empilhado)

# Prepara os dados
df_bar_stacked = dataset.loc[:200,['AMT_INCOME_TOTAL', 'NAME_INCOME_TYPE', 'NAME_FAMILY_STATUS']]

# Gráfico
grafico4 = page.add('df_bar_stacked', ui.plot_card(box = '1 7 9 4',
```

```
Rendimentos Por Tipo de Rendimento e Status Familiar',
df_bar_stacked.columns.tolist(), rows = df_bar_stacked.values.tolist()),

[ui.mark(type = 'interval',

                                x = '=NAME_INCOME_TYPE',

                                y = '=AMT_INCOME_TOTAL',

                                color = '=NAME_FAMILY_STATUS',

                                stack = 'auto')))))

#####

# Gráfico de Total de Rendimentos Por Tipo de Empréstimo (Gráfico de Linha)

# Prepara os dados
df_line = dataset.loc[:200,['SK_ID_CURR', 'AMT_INCOME_TOTAL']]

# Gráfico
grafico5 = page.add('df_line', ui.plot_card(box = '1 11 5 4',
                                title = 'Total de Rendimentos Por Tipo de Empréstimo',

                                data =

                                plot = ui.plot(marks =

                                x = '=SK_ID_CURR',

                                y = '=AMT_INCOME_TOTAL',

                                curve = 'smooth')))))

#####

# Gráfico de Total de Rendimentos Por Tipo de Rendimento e Estado Civil (Gráfico de Área)

# Prepara os dados
df_area = dataset.loc[:200,['AMT_INCOME_TOTAL','NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS']]

# Gráfico
grafico6 = page.add('df_area', ui.plot_card(box = '6 11 4 4',

                                title = 'Total de Rendimentos Por Tipo de

                                data = data(fields =

                                plot = ui.plot(marks = [ui.mark(type =

                                x = '=NAME_EDUCATION_TYPE',

                                y = '=AMT_INCOME_TOTAL',

                                color = '=NAME_FAMILY_STATUS')))))

#####

# Gráfico de Total de Rendimentos Por Tipo de Empréstimo (Gráfico de Linha Step)

# Prepara os dados
df_line_step = dataset.loc[:100,['SK_ID_CURR', 'AMT_INCOME_TOTAL']]

# Gráfico
grafico7 = page.add('df_line_step', ui.plot_card(box = '1 15 5 4',
                                title = 'Total de Rendimentos Por Tipo de Empréstimo (Step)',

                                data = data(fields =

                                plot=ui.plot([ui.mark(type = 'path',

                                x

                                = '=SK_ID_CURR',

                                y

                                = '=AMT_INCOME_TOTAL',

                                curve = 'step' )))))
```

```
#####

# Gráfico de Total de Rendimentos Por Tipo de Residência (Gráfico de Barras)

# Prepara os dados
df_b = dataset.loc[:200,['AMT_INCOME_TOTAL', 'NAME_HOUSING_TYPE']]

# Gráfico
grafico8 = page.add(df_b, ui.plot_card(box = '6 15 4 4',
                                     title = 'Total de Rendimentos Por
                                     Tipo de Residência',
                                     data = data(fields =
                                     df_b.columns.tolist(), rows = df_b.values.tolist()),
                                     plot = ui.plot([ui.mark(type =
                                     'interval',
                                     x =
                                     'NAME_HOUSING_TYPE',
                                     y =
                                     'AMT_INCOME_TOTAL' )]))))

page.save()

# Fim
```

9.6. Visualização de Dados com R

Função Plot e Paleta de Cores

```
# Func Plot e Paleta de Cores

# Diretório de trabalho
# Altere o diretório abaixo de acordo com o diretório na sua máquina
setwd("~/Dropbox/DSA/Data Viz/Cap06/01-FuncPlot")
# Windows -> setwd("C:/DSA/Data Viz/Cap06/01-FuncPlot")
getwd()

# Criando um plot básico
x <- rnorm(1000)
?plot
plot(x)
plot(x, col = "red")
colors()

# Carregando dados
vendas <- read.csv("vendas.csv", header = TRUE)

# Especificando o tipo de plot - l para linha
plot(vendas$units ~ as.Date(vendas$date, "%d/%m/%y"), type = "l", col = "blue")

# Definindo as cores de fundo
?par
par(bg = "gray")
plot(x)

# Gerando contraste entre a área de desenho e a área do gráfico
x <- rnorm(100)
plot(x, type = "n")
x <- par("usr")
rect(x[1],x[3],x[2],x[4], col = "lightgray")
points(x)

# Definindo as elementos de texto: axis labels, titles, plot titles e legends
plot(x, main = "Titulo do Plot", col.axis = "blue", col.lab = "red", col.main = "darkblue", xlab = "Mês", ylab = "Vendas")

# Formatando
par(col.axis = "black", col.lab = "#444444", col.main = "darkblue")
plot(x)

# Adicionando título
par(col.axis = "black", col.lab = "#444444", col.main = "darkblue")
plot(x, col.axis = "blue", col.lab = "red", col.main = "darkblue", xlab = "Mês", ylab = "Vendas")
title("Vendas de 2010", col.main = "blue")

# Selecionando a combinação de cores e paletas
?palette
palette(c("red", "blue", "green", "orange"))
palette("default")

# Pacote RColorBrewer
install.packages("RColorBrewer")
library(RColorBrewer)

# Visualizando a paleta de cores
display.brewer.all()
brewer.pal(7, "YlOrRd")
display.brewer.pal(7, "YlOrRd")
palette(brewer.pal(7, "YlOrRd"))
pal1 <- brewer.pal(7, "YlOrRd")

# Definindo as fontes e tamanho
par(family = "serif", font = 2)
names(pdfFonts())

# Definindo o sistema de símbolos do plot e tamanhos
chuvas <- read.csv("chuvas.csv")
View(chuvas)
```

```
plot(chuvas$Tokyo, ylim = c(0,250), main = "Chuva Média por Mês", xlab = "Mês", ylab = "Chuvas(mm)", pch = 1)
```

```
# Acrescentando outros dados ao gráficos
points(chuvas$NewYork, pch = 2)
points(chuvas$London, pch = 3)
points(chuvas$Berlin, pch = 4)
```

```
# Legenda
?legend
legend("top", legend = c("Tokyo", "New York", "London", "Berlin"), ncol = 4, cex = 0.8, bty = "n", pch = 1:4)
```

```
# Escolhendo estilos de linha e largura
plot(chuvas$Tokyo,
      ylim = c(0,250),
      main = "Chuva Média por Mês",
      xlab = "Mês",
      ylab = "Chuvas(mm)",
      type = "l",
      lty = 1,
      lwd = 2)
```

```
# Acrescentando linhas
lines(chuvas$NewYork, lty = 2, lwd = 2)
lines(chuvas$London, lty = 3, lwd = 2)
lines(chuvas$Berlin, lty = 4, lwd = 2)
```

```
# Legenda
legend("top", legend = c("Tokyo", "New York", "London", "Berlin"), ncol = 4, cex = 0.8, bty = "n", lty = 1:4, lwd = 2)
```

```
# Definindo estilo da área do gráfico
x <- morm(100)
par(bty = "l")
plot(x)
```

```
# Adicionando contorno
par(oma = c(1,1,1,1))
plot(x, bty = "l")
box(which = "figure")
```

Gerando Plots Padrões (Histogramas Boxplots e Scatterplots)

```
# Gerando Plots Padrões (histogramas, boxplots e scatterplots)
```

```
# Diretório de trabalho
# Altere o diretório abaixo de acordo com o diretório na sua máquina
setwd("~/Dropbox/DSA/Data Viz/Cap06/02-Plots-Padrees")
getwd()
```

```
# Carregando o dataset
auto <- read.csv("carros.csv")
View(auto)
str(auto)
```

```
# Convertendo uma das variáveis para o tipo fator e anexando o dataset
auto$cylinders <- factor(auto$cylinders, levels = c(3,4,5,6,8), labels = c("3cyl", "4cyl", "5cyl", "6cyl", "8cyl"))
attach(auto)
str(auto)
```

```
# Histograma simples
hist(acceleration)
```

```
# Histograma em cor azul
hist(acceleration, col = "blue", xlab = "acceleration", main = "Histograma de Aceleração", breaks = 15)
```

```
# Histograma Colorido
hist(mpg, col = rainbow(12))
```

```
# Histograma com linha
hist(mpg, prob = TRUE)
lines(density(mpg))
```

```
# Boxplot
boxplot(mpg, xlab = "Milhas por Galão")
```

```

# Boxplot
boxplot(mpg ~ model_year, xlab = "Milhas por Galão")

# Plot
plot(mpg ~ horsepower)

# Pair Plot
pairs(~mpg+displacement+horsepower+weight)

# Plot com linha de regressão
plot(mpg ~ horsepower)
reg <- lm(mpg ~ horsepower)
abline(reg)

# Preenchendo um plot vazio
plot(mpg ~ horsepower, type = "n")
with(subset(auto, cylinders == "8cyl"), points(horsepower, mpg, col = "blue"))
with(subset(auto, cylinders == "6cyl"), points(horsepower, mpg, col = "red"))
with(subset(auto, cylinders == "5cyl"), points(horsepower, mpg, col = "yellow"))
with(subset(auto, cylinders == "4cyl"), points(horsepower, mpg, col = "green"))
with(subset(auto, cylinders == "3cyl"), points(horsepower, mpg))

# Selecionando o dispositivo gráfico

# Post Script
postscript(file = "auto-scatter.ps")
boxplot(mpg)
dev.off()

# Auto scatter
pdf(file = "auto-scatter.pdf")
boxplot(mpg)
dev.off()

```

ggplot2

```

# Criando Gráficos com ggplot2

# Diretório de trabalho
# Altere o diretório abaixo de acordo com o diretório na sua máquina
setwd("~/Dropbox/DSA/Data Viz/Cap06/03-ggplot2")
getwd()

# Instalando e carregando o pacote
install.packages("ggplot2")
library(ggplot2)

# Definindo o conjunto de dados com dataset tips
??tips
data(tips, package = 'reshape2')
View(tips)

# Camada 1
?aes
??aes
camada1 <- geom_point(
  mapping = aes(x = total_bill, y = tip, color = sex),
  data = tips,
  size = 3
)

ggplot() + camada1

# Construindo um modelo de regressão
modelo_base <- lm(tip ~ total_bill, data = tips)
modelo_fit <- data.frame(
  total_bill = tips$total_bill,
  predict(modelo_base, interval = "confidence")
)

head(modelo_fit)

```



```

# Camada 2
camada2 <- geom_line(
  mapping = aes(x = total_bill, y = fit),
  data = modelo_fit,
  color = "darkred"
)

ggplot() + camada1 + camada2

# Camada 3
camada3 <- geom_ribbon(
  mapping = aes(x = total_bill, ymin = lwr, ymax = upr),
  data = modelo_fit,
  alpha = 0.3
)

ggplot() + camada1 + camada2 + camada3

# Versão final otimizada
ggplot(tips, aes(x = total_bill, y = tip)) +
  geom_point(aes(color = sex)) +
  geom_smooth(method = 'lm')

# Gravando o gráfico em um objeto
myplot <- ggplot(tips, aes(x = total_bill, y = tip)) +
  geom_point(aes(color = sex)) +
  geom_smooth(method = 'lm')

class(myplot)
print(myplot)

# Colocando gráficos lado a lado na área de desenho

##### Usando Base Plotting System #####
# Carregando os dados
bikes <- read.csv("bikes.csv")
head(bikes)
str(bikes)
bikes$season <- factor(bikes$season, levels = c(1,2,3,4), labels = c("Primavera", "Verão", "Outono", "Inverno"))
attach(bikes)
head(bikes$season)

# Dividindo a área de desenho em 4 sub-áreas
par(mfrow = c(2,2))

# Coletando amostras dos dados
primavera <- subset(bikes, season == "Primavera")$cnt
verao <- subset(bikes, season == "Verão")$cnt
outono <- subset(bikes, season == "Outono")$cnt
inverno <- subset(bikes, season == "Inverno")$cnt

# Desenhando os gráficos
hist(primavera, prob = TRUE, xlab = "Aluguel Diário na Primavera", main = "")
lines(density(primavera))

hist(verao, prob = TRUE, xlab = "Aluguel Diário no Verão", main = "")
lines(density(verao))

hist(outono, prob = TRUE, xlab = "Aluguel Diário no Outono", main = "")
lines(density(outono))

hist(inverno, prob = TRUE, xlab = "Aluguel Diário no Inverno", main = "")
lines(density(inverno))

##### Usando ggplot2 #####
qplot(cnt, data = bikes) + facet_wrap(~ season, nrow = 2) + geom_histogram(fill = "blue")
qplot(cnt, data = bikes, fill = season)

# Gráficos lado a lado na mesma área de gráfico
qplot(season, cnt, data = bikes, geom = c("boxplot"), fill = season)
ggplot(bikes, aes(x = season, y = cnt)) + geom_boxplot()

```

```

# Plots multivariados
bikes <- read.csv("bikes.csv")
bikes$season <- factor(bikes$season, levels = c(1,2,3,4), labels = c("Primavera", "Verão", "Outono", "Inverno"))
bikes$weathersit <- factor(bikes$weathersit, levels = c(1,2,3), labels = c("Sol", "Nublado", "Chuva"))
bikes$windspeed.fac <- cut(bikes$windspeed, breaks = 3, labels = c("Baixo", "Médio", "Alto"))
bikes$weekday <- factor(bikes$weekday, levels = c(0:6), labels = c("Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sab"))
attach(bikes)

# Criando o objeto plot
plot <- ggplot(bikes, aes(temp, cnt))

# Adicionando camadas ao plot
plot + geom_point(size = 3,
  aes(color = factor(windspeed.fac))) +
  geom_smooth(method = "lm", se = FALSE, col = "red") +
  facet_grid(weekday ~ season) +
  theme(legend.position = "bottom")

# Scatter Plot e Scatter Plot 3d

# Scatter Plot

# Dados
data = data.frame(cond = rep(c("Obs 1", "Obs 2"), each = 10), var1 = 1:100 + rnorm(100, sd = 9), var2 = 1:100 + rnorm(100, sd = 16))
head(data)

# Plot
ggplot(data, aes(x = var1, y = var2)) +
  geom_point(shape = 1) +
  geom_smooth(method = lm, color = "red", se = FALSE)

# Scatter Plot 3D
install.packages("scatterplot3d")
library(scatterplot3d)

# Definindo o tamanho da área de desenho
par(mfrow = c(1,1))

scatterplot3d(x = mtcars$wt,
  y = mtcars$displ,
  z = mtcars$mpg)

scatterplot3d(mtcars$wt, mtcars$displ, mtcars$mpg,
  pch=16, highlight.3d = TRUE, angle = 20,
  xlab = "Peso", ylab = "Deslocamento", zlab = "Consumo de Combustível(mpg)",
  type = "h",
  main = "Relacionamento Entre Características de Automóveis")

# BarPlot, Histograma e Polígono de Frequência

# Bar Plot

# Dados
data = data.frame(grupo = c("A ", "B ", "C ", "D ") ,
  valor = c(33,62,56,67) ,
  num_obs = c(100,500,459,342))

# Gerando a massa de dados
data$right = cumsum(data$num_obs) + 30 * c(0:(nrow(data)-1))
data$left = data$right - data$num_obs

# Plot
ggplot(data, aes(ymin = 0)) +
  geom_rect(aes(xmin = left, xmax = right,
    ymax = valor, colour = grupo, fill = grupo)) +
  xlab("Número de obs") + ylab("Valor")

# Histograma
ggplot(diamonds, aes(carat)) +
  geom_histogram()

ggplot(diamonds, aes(carat)) +
  geom_histogram(binwidth = 0.01)

```

```

ggplot(diamonds, aes(carat)) +
  geom_histogram(bins = 200)

ggplot(diamonds, aes(price, fill = cut)) +
  geom_histogram(binwidth = 500)

# Polígono de Frequência
ggplot(diamonds, aes(price, colour = cut)) +
  geom_freqpoly(binwidth = 500)

# Para facilitar a comparação de distribuições com contagens muito diferentes, colocamos densidade no eixo y
# em vez da contagem padrão
ggplot(diamonds, aes(price, ..density.., colour = cut)) +
  geom_freqpoly(binwidth = 500)

# Personalizando o Gráfico

# Dados
head(mtcars)

# Plot simples
ggplot(data = mtcars, aes(x = disp, y = mpg)) + geom_point()

# Outro aspecto que pode ser mapeado nesse gráfico é a cor dos pontos
ggplot(data = mtcars,
  aes(x = disp, y = mpg,
    colour = as.factor(am))) + geom_point()

# No entanto, também podemos mapear uma variável contínua à cor dos pontos:
ggplot(mtcars, aes(x = disp, y = mpg, colour = cyl)) + geom_point()

# Também podemos mapear o tamanho dos pontos à uma variável de interesse:
# A legenda é inserida no gráfico automaticamente
ggplot(mtcars, aes(x = disp, y = mpg, colour = cyl, size = wt)) + geom_point()

# Os geoms definem qual forma geométrica será utilizada para a visualização dos dados no gráfico.
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg)) + geom_boxplot()

# Histogramas
ggplot(mtcars, aes(x = mpg), binwidth = 30) + geom_histogram()

# Gráfico de Barras
ggplot(mtcars, aes(x = as.factor(cyl))) + geom_bar()

# Personalizando os gráficos
colors()

ggplot(mtcars, aes(x = as.factor(cyl), y = mpg,
  colour = as.factor(cyl))) + geom_boxplot()

ggplot(mtcars, aes(x = as.factor(cyl), y = mpg,
  fill = as.factor(cyl))) + geom_boxplot()

ggplot(mtcars,
  aes(x = as.factor(cyl), y = mpg)) +
  geom_boxplot(color = "blue", fill = "seagreen4")

# Podemos alterar os eixos
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram() +
  xlab("Milhas por galão") + ylab("Frequência")

# Alterar os limites do gráfico
ggplot(mtcars, aes(x = mpg)) + geom_histogram() + xlab("Milhas por galão") + ylab("Frequência") +
  xlim(c(0, 40)) +
  ylim(c(0,8))

# Legendas
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +
  labs(fill = "cyl")

# Trocando a posição da legenda
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +

```

```

labs(fill = "cyl") +
theme(legend.position="top")

# Sem legenda
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +
  guides(fill=FALSE)

# Facets
ggplot(mtcars, aes(x = mpg, y = disp, colour = as.factor(cyl))) +
  geom_point() +
  facet_grid(am~.)

ggplot(mtcars, aes(x = mpg, y = disp, colour = as.factor(cyl))) +
  geom_point() +
  facet_grid(.~am)

# Plots diferentes juntos (diferente de Facet)
library(gridExtra)
library(ggplot2)

# Dataset diamonds
data(diamonds)

# Histograma como plot1
plot1 <- qplot(price, data = diamonds, binwidth = 1000)

# ScatterPlot como plot2
plot2 <- qplot(carat, price, data = diamonds, colour = cut)

# Combina os 2 plots na mesma área
grid.arrange(plot1, plot2, ncol = 1)

# Facets com reshape
library(reshape2)
library(plotly)

sp <- ggplot(tips, aes(x=total_bill, y=tip/total_bill)) + geom_point(shape=1)
sp + facet_grid(sex ~ .)
ggplotly()
sp + facet_grid(. ~ sex)
ggplotly()
sp + facet_wrap( ~ day, ncol = 2)
ggplotly()

ggplot(mpg, aes(displ, hwy)) + geom_point() + facet_wrap(~manufacturer)
ggplotly()

```

Heat Maps

```

# Heat Maps

# Diretório de trabalho
# Altere o diretório abaixo de acordo com o diretório na sua máquina
setwd("~/Dropbox/DSA/DataViz/Cap06/04-Heatmaps")
getwd()

# Pacote
install.packages("RColorBrewer")
library(RColorBrewer)

# Carregando os dados
vendas <- read.csv("vendas.csv")
head(vendas)

# Preparando os dados
rownames(vendas) <- vendas[,1]
vendas <- vendas[,-1]
data_matrix <- data.matrix(vendas)

```

```

head(vendas)

# Paleta de cores e tamanho dos bins (retângulos)
pal = brewer.pal(7,"YlOrRd")
breaks <- seq(3000,12000,1500)

# Cria layout com 1 linha e 2 colunas (para o heatmap e escala); A coluna heatmap é 8 vezes maior que a coluna da escala
?layout
layout(matrix(data = c(1,2), nrow = 1, ncol = 2), widths = c(8,1), heights = c(1,1))

# Escolha margens para o heat map
par(mar = c(5,10,4,2), oma = c(0.2,0.2,0.2,0.2), mex = 0.5)

# Cria o gráfico
?image
image(x = 1:nrow(data_matrix), y = 1:ncol(data_matrix),
      z = data_matrix,
      axes = FALSE,
      xlab = "Mês",
      ylab = "",
      col = pal[1:(length(breaks)-1)],
      breaks = breaks,
      main = "Heat Map de Vendas")

# Adiciona labels ao eixo x
axis(1, at=1:nrow(data_matrix), labels = rownames(data_matrix), col = "white", las = 1)

# Adiciona labels ao eixo y
axis(2, at=1:ncol(data_matrix), labels = colnames(data_matrix), col = "white", las = 1)

# Adiciona linhas divisórias
abline(h=c(1:ncol(data_matrix))+0.5,
       v=c(1:nrow(data_matrix))+0.5, col="white",lwd=2,xpd=FALSE)

# Adicionando breaks para usar como escala
breaks2 <- breaks[-length(breaks)]

# Color Scale
par(mar = c(5,1,4,7))

# Se você obtiver um erro de margens de figura ao executar o código acima, amplie o dispositivo de plotagem ou
# ajuste as margens para que o gráfico e a escala se encaixem no dispositivo.
image(x=1, y=0:length(breaks2),z=t(matrix(breaks2))*1.001,
     col=pal[1:length(breaks)-1],
     axes=FALSE,
     breaks = breaks,
     xlab = "",
     ylab = "",
     xaxt = "n")

# Labels e linhas divisórias
axis(4, at=0:(length(breaks2)-1), labels = breaks2, col = "white", las = 1)
abline(h = c(1:length(breaks2)), col = "white", lwd = 2, xpd = F)

# Correlation Heat Maps

# Carrega os dados
genes <- read.csv("genes.csv")
head(genes)

# Prepara os dados
rownames(genes) <- genes[,1]
data_matrix <- data.matrix(genes[,-1])

# Cores e breaks
pal = heat.colors(5)
breaks <- seq(0,1,0.2)

# Layout
layout(matrix(data = c(1,2), nrow = 1, ncol = 2), widths = c(8,1), heights = c(1,1))
par(mar = c(3,7,12,2), oma = c(0.2,0.2,0.2,0.2), mex = 0.5)

# Cria o gráfico
image(x=1:nrow(data_matrix),y=1:ncol(data_matrix),
     z = data_matrix,
     xlab = "",
     ylab = "",

```

```

    breaks = breaks,
    col = pal,
    axes = FALSE)

# Adiciona texto
text(x=1:nrow(data_matrix)+0.75, y=par("usr")[4] + 1.25, srt = 45, adj = 1, labels = rownames(data_matrix), xpd = TRUE)

# Labels
axis(2,at=1:ncol(data_matrix), labels = colnames(data_matrix), col = "white", las = 1)

# Linhas
abline(h = c(1:ncol(data_matrix))+0.5, v = c(1:nrow(data_matrix))+0.5, col = "white", lwd = 2, xpd = F)

# Título
title("Correlação Entre Genes", line = 8, adj = 0)


# Sumarizando Dados Multivariados em um único Heat Map

# Carregando os dados
nba <- read.csv("nba.csv")
head(nba)

# Pacote de cores
library(RColorBrewer)

# Obtendo o nome das linhas
rownames(nba) <- nba[,1]

# Ajustando os dados
data_matrix <- t(scale(data.matrix(nba[, -1])))

# Paleta de cores
pal = brewer.pal(6, "Blues")

# Vetor com o nome das estatísticas
statnames <- c("Games Played", "Minutes Played", "Total Points", "Field Goals Made", "Field Goals Attempted", "Field Goal Percentage", "Free
Throws Made", "Free Throws Attempted", "Free Throw Percentage", "Three Pointers Made", "Three Pointers Attempted", "Three Point
Percentage", "Offensive Rebounds", "Defensive Rebounds", "Total Rebounds", "Assists", "Steals", "Blocks", "Turnovers", "Fouls")

# Parâmetros do gráfico
par(mar = c(3,14,19,2), oma=c(0.2,0.2,0.2,0.2), mex=0.5)

# Heat map
image(x = 1:nrow(data_matrix),
      y = 1:ncol(data_matrix),
      z = data_matrix,
      xlab = "",
      ylab = "",
      col = pal,
      axes = FALSE)

# Label no eixo x
text(1:nrow(data_matrix), par("usr")[4] + 1, srt = 45, adj = 0, labels = statnames, xpd = TRUE, cex = 0.85)

# Label no eixo y
axis(side=2, at = 1:ncol(data_matrix), labels = colnames(data_matrix), col = "white", las = 1, cex.axis = 0.85)

# Linha divisória
abline(h = c(1:ncol(data_matrix))+0.5, v=c(1:nrow(data_matrix))+0.5, col = "white", lwd = 1, xpd = F)

# Título
text(par("usr")[1]+5, par("usr")[4] + 12, "Performance por Jogo dos 50 Melhores Atletas da NBA", xpd = TRUE, font = 2, cex = 1.5)


# Contour plots
??countour
?volcano
contour(x = 10*1:nrow(volcano),
        y = 10*1:ncol(volcano),
        z = volcano,
        xlab = "Metros a Oeste",
        ylab = "Metros ao Norte",
        main = "Topografia do Vulcão Maunga Whau")

# Criando a área de plotagem vazia

```

```

par(las = 1)
plot(0,0,
      xlim = c(0,10*nrow(volcano)),
      ylim = c(0,10*ncol(volcano)),
      type = "n",
      xlab = "Metros a Oeste",
      ylab = "Metros ao Norte",
      main = "Topografia do Vulcão Maunga Whau")

# Ajustando a área de plotagem e alterando a cor de fundo
u <- par("usr")
rect(u[1],u[3],u[2],u[4], col = "lightgreen")

# Criando o gráfico
contour(x = 10*1:nrow(volcano),
        y = 10*1:ncol(volcano),
        volcano,
        col = "red",
        add = TRUE)

# Preenchendo o Countour Plot (filled contour plots)
filled.contour(x = 10*1:nrow(volcano),
               y = 10*1:ncol(volcano),
               z = volcano,
               color.palette = terrain.colors,
               plot.title = title(main = "Topografia do Vulcão Maunga Whau", xlab = "Metros ao Norte", ylab = "Metros a Oeste"),
               plot.axes = {axis(1, seq(100, 800, by = 100))
                           axis(2, seq(100, 600, by = 100))},
               key.title = title(main="Altura\n(metros)"),
               key.axes = axis(4, seq(90, 190, by = 10)))

# Preenchendo o Countour Plot (filled contour plots) e aumentando o nível de detalhes
filled.contour(x = 10*1:nrow(volcano),
               y = 10*1:ncol(volcano),
               z = volcano,
               color.palette = terrain.colors,
               plot.title = title(main = "Topografia do Vulcão Maunga Whau", xlab = "Metros ao Norte", ylab = "Metros a Oeste"),
               nlevels = 100,
               plot.axes = {axis(1, seq(100, 800, by = 100))
                           axis(2, seq(100, 600, by = 100))},
               key.title = title(main = "Altura\n(metros)"),
               key.axes = axis(4, seq(90, 190, by = 10)))

# Surface plots 3D

# Pacote
install.packages("rgl")
library(rgl)

# Variáveis
z <- 2 * volcano
x <- 10 * (1:nrow(z))
y <- 10 * (1:ncol(z))

# Definindo os limites
zlim <- range(z)
zlen <- zlim[2] - zlim[1] + 1

# Cores
colorlut <- terrain.colors(zlen)
col <- colorlut[ z-zlim[1]+1 ]

# Cria o gráfico
rgl.open()
rgl.surface(x, y, z, color = col, back = "lines")

# Visualizando Séries Temporais com Calendar Heat Maps

# Importa a biblioteca calendarHeat.R
source("calendarHeat.R")

# Dados
stock.data <- read.csv("google.csv")

```

```

# Pacote
install.packages("chron")
library(chron)

# Cria o calendário
calendarHeat(dates = stock.data$Date, values = stock.data$Adj.Close, varname = "Google Adjusted Close")

# Usando o openair
install.packages("openair")
library(openair)

# Usando o CalendarPlot
?calendarPlot
calendarPlot(mydata)

# Gerando massa de dados
mydata$sales <- rnorm(length(mydata$nox), mean = 1000, sd = 1500)

# CalendarPlot
calendarPlot(mydata, pollutant = "sales", main = "Vendas Diárias em 2003")


# Acesse o site e crie sua conta
# https://plot.ly/
# https://plot.ly/r/heatmaps/
# acessando config api
# https://plot.ly/settings/api

# Carregando o pacote
library(plotly)
packageVersion('plotly')

#### TROCANDO CONFIGURACAO DA API
#py <- plotly("RgraphingAPI", "ektgzomjbx")
Sys.setenv("plotly_username"=" user_name ")
Sys.setenv("plotly_api_key"=" key_api ")

#### TROCANDO FUNÇÃO
p <- plot_ly(z = volcano, type = "heatmap")
api_create(p)

#### TROCANDO FUNÇÃO
#### TROCANDO filename
chart_link = api_create(p, filename = "heatmap_simple")
chart_link

```

```

#####
#           Calendar Heatmap           #
#           by                         #
#           Paul Bleicher              #
# an R version of a graphic from:      #
# http://stat-computing.org/dataexpo/2009/posters/wicklin-allison.pdf    #
# requires lattice, chron, grid packages      #
#####

```

```

## calendarHeat: An R function to display time-series data as a calendar heatmap
## Copyright 2009 Humedica. All rights reserved.

```

```

## This program is free software; you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation; either version 2 of the License, or
## (at your option) any later version.

```

```

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

```

```

## You can find a copy of the GNU General Public License, Version 2 at:
## http://www.gnu.org/licenses/gpl-2.0.html

```

```

calendarHeat <- function(dates,
  values,
  ncolors=99,
  color="r2g",
  varname="Values",

```



```

        date.form = "%Y-%m-%d", ...) {
require(lattice)
require(grid)
require(chron)
if (class(dates) == "character" | class(dates) == "factor" ) {
  dates <- strptime(dates, date.form)
}
caldat <- data.frame(value = values, dates = dates)
min.date <- as.Date(paste(format(min(dates), "%Y"),
  "-1-1", sep = ""))
max.date <- as.Date(paste(format(max(dates), "%Y"),
  "-12-31", sep = ""))
dates.f <- data.frame(date.seq = seq(min.date, max.date, by="days"))

# Merge moves data by one day, avoid
caldat <- data.frame(date.seq = seq(min.date, max.date, by="days"), value = NA)
dates <- as.Date(dates)
caldat$value[match(dates, caldat$date.seq)] <- values

caldat$dotw <- as.numeric(format(caldat$date.seq, "%w"))
caldat$woty <- as.numeric(format(caldat$date.seq, "%U")) + 1
caldat$yr <- as.factor(format(caldat$date.seq, "%Y"))
caldat$month <- as.numeric(format(caldat$date.seq, "%m"))
yrs <- as.character(unique(caldat$yr))
d.loc <- as.numeric()
for (m in min(yrs):max(yrs)) {
  d.subset <- which(caldat$yr == m)
  sub.seq <- seq(1,length(d.subset))
  d.loc <- c(d.loc, sub.seq)
}
caldat <- cbind(caldat, seq=d.loc)

#color styles
r2b <- c("#0571B0", "#92C5DE", "#F7F7F7", "#F4A582", "#CA0020") #red to blue
r2g <- c("#D61818", "#FFAE63", "#FFFFBD", "#B5E384") #red to green
w2b <- c("#045A8D", "#2B8CBE", "#74A9CF", "#BDC9E1", "#F1EEF6") #white to blue

assign("col.sty", get(color))
calendar.pal <- colorRampPalette((col.sty), space = "Lab")
def.theme <- lattice.getOption("default.theme")
cal.theme <-
  function() {
    theme <-
      list(
        strip.background = list(col = "transparent"),
        strip.border = list(col = "transparent"),
        axis.line = list(col="transparent"),
        par.strip.text=list(cex=0.8))
  }
lattice.options(default.theme = cal.theme)
yrs <- (unique(caldat$yr))
nyr <- length(yrs)
print(cal.plot <- levelplot(value~woty*dotw | yr, data=caldat,
  as.table=TRUE,
  aspect=.12,
  layout = c(1, nyr%%7),
  between = list(x=0, y=c(1,1)),
  strip=TRUE,
  main = paste("Calendar Heat Map of ", varname, sep = ""),
  scales = list(
    x = list(
      at= c(seq(2.9, 52, by=4.42)),
      labels = month.abb,
      alternating = c(1, rep(0, (nyr-1))),
      tck=0,
      cex = 0.7),
    y=list(
      at = c(0, 1, 2, 3, 4, 5, 6),
      labels = c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday"),
      alternating = 1,
      cex = 0.6,
      tck=0)),
    xlim =c(0.4, 54.6),
    ylim=c(6.6,-0.6),
    cuts= ncolors - 1,
    col.regions = (calendar.pal(ncolors)),
    xlab="",

```

```

ylab="",
colorkey= list(col = calendar.pal(ncolors), width = 0.6, height = 0.5),
subscripts=TRUE
))
panel.locs <- trellis.currentLayout()
for (row in 1:nrow(panel.locs)) {
  for (column in 1:ncol(panel.locs)) {
    if (panel.locs[row, column] > 0)
  {
    trellis.focus("panel", row = row, column = column,
      highlight = FALSE)
xyetc <- trellis.panelArgs()
subs <- caldat[xyetc$subscripts,]
dates.fsubs <- caldat[caldat$yr == unique(subs$yr),]
y.start <- dates.fsubs$dotw[1]
y.end <- dates.fsubs$dotw[nrow(dates.fsubs)]
dates.len <- nrow(dates.fsubs)
adj.start <- dates.fsubs$woty[1]

for (k in 0:6) {
  if (k < y.start) {
    x.start <- adj.start + 0.5
  } else {
    x.start <- adj.start - 0.5
  }
  if (k > y.end) {
    x.finis <- dates.fsubs$woty[nrow(dates.fsubs)] - 0.5
  } else {
    x.finis <- dates.fsubs$woty[nrow(dates.fsubs)] + 0.5
  }
  grid.lines(x = c(x.start, x.finis), y = c(k - 0.5, k - 0.5),
    default.units = "native", gp=gpar(col = "grey", lwd = 1))
}
if (adj.start < 2) {
  grid.lines(x = c(0.5, 0.5), y = c(6.5, y.start - 0.5),
    default.units = "native", gp=gpar(col = "grey", lwd = 1))
  grid.lines(x = c(1.5, 1.5), y = c(6.5, -0.5), default.units = "native",
    gp=gpar(col = "grey", lwd = 1))
  grid.lines(x = c(x.finis, x.finis),
    y = c(dates.fsubs$dotw[dates.len] - 0.5, -0.5), default.units = "native",
    gp=gpar(col = "grey", lwd = 1))
  if (dates.fsubs$dotw[dates.len] != 6) {
    grid.lines(x = c(x.finis + 1, x.finis + 1),
      y = c(dates.fsubs$dotw[dates.len] - 0.5, -0.5), default.units = "native",
      gp=gpar(col = "grey", lwd = 1))
  }
  grid.lines(x = c(x.finis, x.finis),
    y = c(dates.fsubs$dotw[dates.len] - 0.5, -0.5), default.units = "native",
    gp=gpar(col = "grey", lwd = 1))
}
for (n in 1:51) {
  grid.lines(x = c(n + 1.5, n + 1.5),
    y = c(-0.5, 6.5), default.units = "native", gp=gpar(col = "grey", lwd = 1))
}
x.start <- adj.start - 0.5

if (y.start > 0) {
  grid.lines(x = c(x.start, x.start + 1),
    y = c(y.start - 0.5, y.start - 0.5), default.units = "native",
    gp=gpar(col = "black", lwd = 1.75))
  grid.lines(x = c(x.start + 1, x.start + 1),
    y = c(y.start - 0.5, -0.5), default.units = "native",
    gp=gpar(col = "black", lwd = 1.75))
  grid.lines(x = c(x.start, x.start),
    y = c(y.start - 0.5, 6.5), default.units = "native",
    gp=gpar(col = "black", lwd = 1.75))
  if (y.end < 6) {
    grid.lines(x = c(x.start + 1, x.finis + 1),
      y = c(-0.5, -0.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
    grid.lines(x = c(x.start, x.finis),
      y = c(6.5, 6.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
  } else {
    grid.lines(x = c(x.start + 1, x.finis),
      y = c(-0.5, -0.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
    grid.lines(x = c(x.start, x.finis),

```

```

    y = c(6.5, 6.5), default.units = "native",
    gp=gpar(col = "black", lwd = 1.75))
  }
  } else {
    grid.lines(x = c(x.start, x.start),
      y = c(- 0.5, 6.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
  }
}

if (y.start == 0 ) {
  if (y.end < 6 ) {
    grid.lines(x = c(x.start, x.finis + 1),
      y = c(-0.5, -0.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
    grid.lines(x = c(x.start, x.finis),
      y = c(6.5, 6.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
  } else {
    grid.lines(x = c(x.start + 1, x.finis),
      y = c(-0.5, -0.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
    grid.lines(x = c(x.start, x.finis),
      y = c(6.5, 6.5), default.units = "native",
      gp=gpar(col = "black", lwd = 1.75))
  }
}

for (j in 1:12) {
  last.month <- max(dates.fsubs$seq[dates.fsubs$month == j])
  x.last.m <- dates.fsubs$woty[last.month] + 0.5
  y.last.m <- dates.fsubs$dotw[last.month] + 0.5
  grid.lines(x = c(x.last.m, x.last.m), y = c(-0.5, y.last.m),
    default.units = "native", gp=gpar(col = "black", lwd = 1.75))
  if ((y.last.m) < 6) {
    grid.lines(x = c(x.last.m, x.last.m - 1), y = c(y.last.m, y.last.m),
      default.units = "native", gp=gpar(col = "black", lwd = 1.75))
    grid.lines(x = c(x.last.m - 1, x.last.m - 1), y = c(y.last.m, 6.5),
      default.units = "native", gp=gpar(col = "black", lwd = 1.75))
  } else {
    grid.lines(x = c(x.last.m, x.last.m), y = c(- 0.5, 6.5),
      default.units = "native", gp=gpar(col = "black", lwd = 1.75))
  }
}
}
}
}
trellis.unfocus()
}
lattice.options(default.theme = def.theme)
}

## Example of use: Plot financial data
## This code is not run.
if(FALSE) {

#create faux data; skip this to use data from a file or stock data
#ndays <- 1500 #set number of days
#dates <- as.POSIXlt(seq(Sys.Date()- ndays, Sys.Date() - 1, by="days"))
#vals <- runif(ndays, -100, 100)

#stock data:
stock <- "MSFT"
start.date <- "2006-01-12"
end.date <- Sys.Date()
quote <- paste("http://ichart.finance.yahoo.com/table.csv?s=",
  stock,
  "&a=", substr(start.date,6,7),
  "&b=", substr(start.date, 9, 10),
  "&c=", substr(start.date, 1,4),
  "&d=", substr(end.date,6,7),
  "&e=", substr(end.date, 9, 10),
  "&f=", substr(end.date, 1,4),
  "&g=d&ignore=.csv", sep="")
stock.data <- read.csv(quote, as.is=TRUE)

# Plot as calendar heatmap
calendarHeat(stock.data$Date,
  stock.data$Adj.Close, varname="MSFT Adjusted Close")
}

```

Criando Gráficos com Lattice

```
# Criando gráficos com lattice

# Instalando o pacote
install.packages("lattice")
library(lattice)

# Carregando o dataset
auto <- read.csv("carros.csv", stringsAsFactors = FALSE)
head(auto)
str(auto)

# Convertendo a coluna cylinders para o tipo fator
cyl.factor <- factor(auto$cylinders, labels = c("3cyl", "4cyl", "5cyl", "6cyl", "8cyl"))

# Criando o gráfico - Scatterplots
# (Outros disponíveis são barchart, bwplot, densityplot, dotplot, histogram, etc.).
?xyplot
xyplot(mpg ~ weight|cyl.factor, data = auto, main = "Scatterplots Por Cilindros", ylab = "Milhas por Galão", xlab = "Peso do Carro")

# Para ver as bases de dados disponíveis
data()

# Para carregar a base de dados
library(MASS)
data(Cars93)
dim(Cars93)
?Cars93
names(Cars93)
head(Cars93)
View(Cars93)

# Mostra os valores únicos contidos em colunas específicas
unique(Cars93$Origin)
unique(Cars93$Make)

# Um dos tipos de gráficos do pacote lattice
xyplot(MPG.city ~ EngineSize, Cars93)

# Selecionando dados
# seleciona dados de quatro fabricantes para trabalhar
data <- subset(Cars93, Manufacturer %in% c('Ford', 'Chevrolet', 'Toyota', 'Honda'))

# Criando o gráfico
xyplot(MPG.city ~ EngineSize | Manufacturer, data)
xyplot(MPG.city ~ EngineSize | Manufacturer, data, type = 'l')

xyplot(MPG.city ~ EngineSize | Manufacturer, data, group = Type, auto.key = T)

# seleciona apenas três tipos de carro
data <- subset(Cars93, Type %in% c('Compact', 'Small', 'Large'))

# Plota de acordo com origem, tipo e por número de passageiros
xyplot(MPG.city ~ EngineSize | Origin + Type, data, group = Passengers, auto.key = list(space = 'right'))

# Legenda
xyplot(MPG.city ~ EngineSize | Manufacturer, data, group = Type, auto.key = list(space = 'right'))

# Título e Labels
xyplot(MPG.city ~ EngineSize | Manufacturer, data,
  group = Type,
  auto.key = list(space = 'right'),
  main = 'Relação entre rendimento e motor',
  xlab = 'tamanho do motor (litros)',
  ylab = 'rendimento (milhas por galão)')

# Separação ao longo do eixo x
xyplot(MPG.city ~ EngineSize | Manufacturer, data, between = list(x = 1))

# Salvando o gráfico
dev.copy(device = pdf, file = "lattice.pdf", width = 600, paper = "USr")
dev.off()

# Box Plots

# Criando o gráfico - Boxplots
```

```
?bwplot
bwplot(~auto$mpg|cyl.factor, main = "Consumo / Número de Cilindros", xlab = "Milhas por Galão")

# Parâmetros do Gráfico
?trellis.par.set
trellis.par.set(theme = col.whitebg())

# Criando o gráfico
bwplot(~mpg|cyl.factor, data = auto, main = "Consumo / Número de Cilindros", xlab = "Milhas por Galão", layout = c(2,3), aspect = 1)

# Violin Plot

# O Violin Plot é semelhante aos boxplots, exceto que eles também mostram a densidade de probabilidade dos dados
# em diferentes valores (no caso mais simples, isso poderia ser um histograma).

# Tipicamente violin plots incluirão um marcador para a mediana dos dados e uma caixa indicando o intervalo interquartil,
# como em boxplots padrões. Um violin plot é mais informativo do que um boxplot simples.
# De fato, enquanto um boxplot mostra apenas estatísticas resumidas, como média / mediana e intervalos interquartis,
# o gráfico de violino mostra a distribuição completa dos dados.

# Carregando dados
# NASA Surface meteorology and Solar Energy (SSE) Release 6.0 Data Set (Jan 2008)
nasafile <- 'https://eosweb.larc.nasa.gov/sse/global/text/global_radiation'
nasa <- read.table(file = nasafile, skip = 13, header = TRUE)
View(nasa)

# Criando o gráfico
bwplot(Ann~cut(Lat, pretty(Lat, 40)),
  data=nasa, subset = (abs(Lat)<60),
  xlab = 'Latitude', ylab = 'G(0) (kWh/m²)',
  horizontal = FALSE,
  panel = function(..., box.ratio) {panel.violin(..., col = "lightblue", varwidth = FALSE, box.ratio = box.ratio)
    panel.bwplot(..., col = 'black',
      cex = 0.8, pch = '|', fill = 'gray', box.ratio = .1)
  },
  par.settings = list(box.rectangle = list(col = 'black'),
    plot.symbol = list(pch = '.', cex = 0.1)),
  scales = list(x = list(rot = 45, cex = 0.5))
)

# Ajustando os dados
x <- paste(names(nasa)[3:14], collapse='+')
formula <- as.formula(paste(x, '~cut(Lat, pretty(Lat, 20))', sep="))

# Criando o gráfico
bwplot(formula, data = nasa, subset = (abs(Lat)<60),
  xlab = 'Latitude', ylab = 'G(0) (kWh/m²)',
  outer = TRUE, as.table = TRUE, horizontal = FALSE,
  col = 'lightblue',
  panel = panel.violin,
  scales = list(x = list(rot = 70, cex = 0.5)))

# Violin Plot com o pacote vioplot

# Pacote
install.packages("vioplot")
library(vioplot)

### CONJUNTO DE DADOS NÃO ACESSÍVEL
# Carregando os dados
#ds = read.csv("http://www.math.smith.edu/r/data/help.csv")
#female = subset(ds, female==1)

#with(female, vioplot(pcs[homeless==0], pcs[homeless==1],
#  # horizontal=TRUE, names=c("non-homeless", "homeless"),
#  # col = "lightblue"))
```

Gráficos com ggvis

```
# Pacote ggvis
# http://ggvis.rstudio.com/
```

```

# Session Info
sessionInfo()

# Diretório de trabalho
setwd("~/Dropbox/DSA/Data Viz/Cap06/06-Mapas")

# Pacotes
install.packages("ggvis")
library(ggvis)
library(dplyr)

# Gráfico de Barras
pressure %>% ggvis(x = ~temperature, y = ~pressure) %>% layer_bars()

# Gráfico de Barras com variáveis categóricas
pressure %>% ggvis(~factor(temperature), ~pressure) %>% layer_bars()

# Gráfico de Barras
mtcars %>% ggvis(x = ~cyl) %>% layer_bars()

# Observe como o gráfico de barras difere de um histograma: um histograma tem compartimentos que abrangem
# intervalos de x, mas layer_bars mostra a contagem em cada valor x exclusivo.
mtcars %>% ggvis(~wt) %>% layer_histograms()
mtcars %>% ggvis(~wt) %>% layer_bars()

# Gráfico de Barras Sem empilhamento - as barras se sobrepõem
hec <- as.data.frame(xtabs(Freq ~ Hair + Eye, HairEyeColor))
hec %>% group_by(Eye) %>%
  ggvis(x = ~Hair, y = ~Freq, fill = ~Eye, fillOpacity := 0.5) %>%
  layer_bars(stack = FALSE) %>%
  scale_nominal("fill",
    domain = c("Brown", "Blue", "Hazel", "Green"),
    range = c("#995522", "#8CCFF", "#999933", "#00CC00"))

# Com empilhamento
hec %>% group_by(Eye) %>%
  ggvis(x = ~Hair, y = ~Freq, fill = ~Eye, fillOpacity := 0.5) %>%
  layer_bars() %>%
  scale_nominal("fill",
    domain = c("Brown", "Blue", "Hazel", "Green"),
    range = c("#995522", "#8CCFF", "#999933", "#00CC00"))

# Empilhamento na direção x em vez de padrão y
hec %>% group_by(Eye) %>%
  ggvis(y = ~Hair, fill = ~Eye, fillOpacity := 0.5) %>%
  compute_stack(stack_var = ~Freq, group_var = ~Hair) %>%
  layer_rects(x = ~stack_lwr_, x2 = ~stack_upr_, height = band()) %>%
  scale_nominal("y", range = "height", padding = 0, points = FALSE) %>%
  scale_nominal("fill",
    domain = c("Brown", "Blue", "Hazel", "Green"),
    range = c("#995522", "#8CCFF", "#999933", "#00CC00"))

# Faça o conjunto de dados com x categórico
mtc <- mtcars
mtc$cyl <- factor(mtc$cyl)
mtc %>% ggvis(~cyl, ~mpg) %>% layer_boxplots()

# Contínuo x: largura de preenchimento de caixas entre valores de dados
mtcars %>% ggvis(~cyl, ~mpg) %>% layer_boxplots()

# A largura de ajuste = 0,5 torna 0,5 de largura no espaço de dados, que é 1/4 do
# distância entre valores de dados neste caso particular.
mtcars %>% ggvis(~cyl, ~mpg) %>% layer_boxplots(width = 0.5)

# Tooltip brush
x_bar <- "x&#772;"
sigma_hat <- "&sigma;&#770;"

brushed_summary <- function(items, session, page_loc, ...) {
  if (nrow(items) == 0) return()

  items$key__ <- NULL
  lines <- Map(function(name, vals) {
    paste0(name, ": ",

```

```

      x_bar, " = ", round(mean(vals), 2), " ",
      sigma_hat, " = ", round(sd(vals), 2)
    )
  }, names(items), items)
html <- paste(unlist(lines), collapse = "<br />\n")

show_tooltip(session, page_loc$r + 5, page_loc$t, html)
}

# Scatter plot com brushing
mtcars %>% ggvis(x = ~wt, y = ~mpg) %>%
  layer_points(size.brush := 400) %>%
  handle_brush(brushed_summary)

# Gráficos de Linha
set.seed(1780)
df <- data.frame(x = runif(12), y = runif(12), z = gl(3, 4))

df %>% ggvis(x = ~x, y = ~y) %>% layer_paths()

# Agrupamento, especificado manualmente
df %>% group_by(z) %>%
  ggvis(x = ~x, y = ~y, stroke = ~z, fill := NA) %>%
  layer_paths() %>%
  layer_points()

# Agrupamento pode acontecer após a chamada ggvis ()
df %>%
  ggvis(x = ~x, y = ~y, stroke = ~z, fill := NA) %>%
  group_by(z) %>%
  layer_paths() %>%
  layer_points()

# Dados ordenados por x
df %>% ggvis(x = ~x, y = ~y) %>%
  arrange(x) %>%
  layer_paths() %>%
  layer_points()

# Dashed lines
dat <- data.frame(x = rep(c(0, 1), 6), g = gl(6, 2))
dat %>% group_by(g) %>%
  ggvis(x = ~x, y = ~g) %>%
  layer_paths(strokeDash = ~g) %>%
  add_axis("y", grid = FALSE) %>%
  add_axis("x", grid = FALSE, title = "", tick_size_major = 0, ticks = 0)

# Mapas interativos com ggvis

# Mapeando os dados
map_data = ggplot2::map_data("state")
head(map_data)

# Agrupando e plotando
map_data %>% select(long, lat, group, order, region) %>%
  group_by(group) %>%
  ggvis(x = ~long, y = ~lat) %>%
  layer_paths(fill = ~region) %>%
  hide_legend("fill") %>%
  handle_click(on_click = function(data, ...) {print(data)})

# https://rud.is/b/2014/12/29/making-static-interactive-maps-with-ggvis-using-ggvis-maps-wshiny/
# http://rpubs.com/hrbrmstr/ggvis-maps
# https://github.com/hrbrmstr/ggvis-maps

```

Construindo Mapas Interativos com ggvis e Shiny

```
library(ggvis)
```

```

library(rgdal)
library(rgeos)
library(magrittr)
library(dplyr)
library(RColorBrewer)
library(data.table)
library(maptools)

# Basic plot of Maine -----

maine <- readOGR("data/maine.geojson", "OGRGeoJSON")

map <- ggplot2::fortify(maine, region="name")

map %>%
  ggvis(~long, ~lat) %>%
  group_by(group, id) %>%
  layer_paths(strokeOpacity:=0.5, stroke="#7f7f7f") %>%
  hide_legend("fill") %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=400, height=600, keep_aspect=TRUE)

# Basic map labeling -----

# extract the geographic centers of Maine (with their labels)

county_centers <- maine %>%
  gCentroid(byid=TRUE) %>%
  data.frame %>%
  cbind(name=maine$name %>% gsub(" County, ME", "", .) )

map %>%
  group_by(group, id) %>%
  ggvis(~long, ~lat) %>%
  layer_paths(strokeWidth:=0.25, stroke="#7f7f7f") %>%
  layer_points(data=county_centers, x=~x, y=~y, size:=8) %>%
  layer_text(data=county_centers,
    x=~x+0.05, y=~y, text=~name,
    baseline="middle", fontSize:=8) %>%
  hide_legend("fill") %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=400, height=600, keep_aspect=TRUE)

# Basic choropleth -----

me_pop <- read.csv("data/me_pop.csv", stringsAsFactors=FALSE)
me_crime <- read.csv("data/me_crime.csv", stringsAsFactors=FALSE)

crime_1k <- me_crime %>%
  filter(year==2013) %>%
  select(1,5:12) %>%
  left_join(me_pop) %>%
  mutate(murder_1k=1000*(murder/population_2010),
    rape_1k=1000*(rape/population_2010),
    robbery_1k=1000*(robbery/population_2010),
    aggravated_assault_1k=1000*(aggravated_assault/population_2010),
    burglary_1k=1000*(burglary/population_2010),
    larceny_1k=1000*(larceny/population_2010),
    motor_vehicle_theft_1k=1000*(motor_vehicle_theft/population_2010),
    arson_1k=1000*(arson/population_2010))

map %<>% mutate(id=gsub(" County, ME", "", id)) %>%
  left_join(crime_1k, by=c("id"="county"))

crime_values <- function(x) {
  if(is.null(x)) return(NULL)
  y <- me_crime %>% filter(year==2013, county==x$id) %>% select(1,5:12)
  sprintf("<table width='100%%'>%s</table>",
    paste0("<tr><td style='text-align:left'>", names(y),
      ":",</td><td style='text-align:right'>", format(y), collapse="</td></tr>"))
}

map %>%
  group_by(group, id) %>%
  ggvis(~long, ~lat) %>%
  layer_paths(fill=input_select(label="Crime:",
    choices=crime_1k %>%

```



```

        select(ends_with("1k")) %>%
        colnames %>% sort,
        id="Crime",
        map=as.name),
        strokeWidth:=0.5, stroke:="white") %>%
scale_numeric("fill", range=c("#bfd3e6", "#8c6bb1", "#4d004b")) %>%
add_tooltip(crime_values, "hover") %>%
add_legend("fill", title="Crime Rate/1K Pop") %>%
hide_axis("x") %>% hide_axis("y") %>%
set_options(width=400, height=600, keep_aspect=TRUE)

# or with a constant domain across all crimes - we "need" to log scale the
# crime rates given the huge spread

map %>%
mutate_each(funs(log1p), ends_with("1k")) %>%
group_by(group, id) %>%
ggvis(~long, ~lat) %>%
layer_paths(fill=input_select(label="Crime:",
                             choices=crime_1k %>%
                             select(ends_with("1k")) %>%
                             colnames %>% sort,
                             id="Crime",
                             map=as.name),
            strokeWidth:=0.5, stroke:="black") %>%
scale_numeric("fill", trans="quantile",
              domain=c(0, crime_1k %>% mutate_each(funs(log1p),
                                                    ends_with("1k")) %>%
                      select(ends_with("1k")) %>% max),
              range=brewer.pal(9, "BuPu")) %>%
add_tooltip(crime_values, "hover") %>%
add_legend("fill", title="Crime Rate/1K Pop") %>%
hide_axis("x") %>% hide_axis("y") %>%
set_options(width=400, height=600, keep_aspect=TRUE)

# US drought choropleth with custom projection -----

us <- readOGR("data/us.geojson", "OGRGeoJSON")
us <- us[!us$STATEFP %in% c("02", "15", "72"),]

us_aea <- spTransform(us, CRS("+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b=6370997 +units=m +no_defs"))

map <- ggplot2::fortify(us_aea, region="GEOID")

droughts <- read.csv("data/dm_export_county_20141223.csv")
droughts$Id <- sprintf("%05d", as.numeric(as.character(droughts$FIPS)))
droughts$total <- with(droughts, (D0+D1+D2+D3+D4)/5)

map_d <- merge(map, droughts, all.x=TRUE)

ramp <- colorRampPalette(c("white", brewer.pal(n=9, name="YlOrRd")), space="Lab")

map_d$fill_col <- as.character(cut(map_d$total, seq(0,100,10), include.lowest=TRUE, labels=ramp(10)))
map_d$fill_col <- ifelse(is.na(map_d$fill_col), "FFFFFF", map_d$fill_col)

drought_values <- function(x) {
  if(is.null(x) | !(x$Id %in% droughts$Id)) return(NULL)
  y <- droughts %>% filter(id==x$Id) %>% select(1,3,4,6:10)
  sprintf("<table width='100%'>%s</table>",
    paste0("<tr><td style='text-align:left'>", names(y),
           ":",</td><td style='text-align:right'>", format(y), collapse="</td></tr>"))
}

map_d %>%
group_by(group, id) %>%
ggvis(~long, ~lat) %>%
layer_paths(fill=~fill_col, strokeOpacity := 0.5, strokeWidth := 0.25) %>%
add_tooltip(drought_values, "hover") %>%
hide_legend("fill") %>%
hide_axis("x") %>% hide_axis("y") %>%
set_options(width=900, height=600, keep_aspect=TRUE)

# World domination -----

world <- readOGR("data/ne_50m_admin_0_countries.geojson", layer="OGRGeoJSON")
world <- world[!world$iso_a3 %in% c("ATA"),]
world <- spTransform(world, CRS("+proj=wintri"))

```

```

map_w <- ggplot2::fortify(world, region="iso_a3")

launch_sites <- rbindlist(lapply(ogrListLayers("data/launch-sites.kml")[c(1:2,4:9)], function(layer) {
  tmp <- readOGR("data/launch-sites.kml", layer)
  places <- data.table(coordinates(tmp)[,1:2], as.character(tmp$Name))
}))

setnames(launch_sites, colnames(launch_sites), c("lon", "lat", "name"))
coordinates(launch_sites) <- ~lon+lat
launch_sites <- as.data.frame(SpatialPointsDataFrame(spTransform(
  SpatialPoints(launch_sites, CRS("+proj=longlat")), CRS("+proj=wintri")),
  launch_sites@data))

map_w %>%
  group_by(group, id) %>%
  ggvis(~long, ~lat) %>%
  layer_paths(fill="#252525", stroke="white", strokeOpacity:=0.5, strokeWidth:=0.25) %>%
  layer_points(data=launch_sites, stroke="white", x=~lon, y=~lat, fill="#cb181d", size:=25, fillOpacity:=0.5, strokeWidth:=0.25) %>%
  hide_legend("fill") %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=900, height=500, keep_aspect=TRUE)

```

```

# This is the server logic for a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

#install.packages("maptools")
library(shiny)
library(ggvis)
library(rgdal)
library(rgeos)
library(magrittr)
library(dplyr)
library(RColorBrewer)
library(data.table)
library(maptools)

shinyServer(function(input, output, session) {

# Show progress bar while loading everything -----

  progress <- shiny::Progress$new()
  progress$set(message="Loading maps/data", value=0)

# First plot -----

  #maine <- readOGR("data/maine.geojson", "OGRGeoJSON")
  ### EXECUTAR NO WINDOWS
  maine <- readOGR("data/maine.geojson")

  map <- ggplot2::fortify(maine, region="name")

  map %>%
    group_by(group, id) %>%
    ggvis(~long, ~lat) %>%
    layer_paths(strokeOpacity:=0.15) %>%
    hide_legend("fill") %>%
    hide_axis("x") %>% hide_axis("y") %>%
    set_options(width=400, height=600, keep_aspect=TRUE) %>%
    bind_shiny("maine")

# Second plot -----

  county_centers <- maine %>%
    gCentroid(byid=TRUE) %>%
    data.frame %>%
    cbind(name=maine$name %>% gsub(" County, ME", "", .) )

  map %>%
    group_by(group, id) %>%
    ggvis(~long, ~lat) %>%
    layer_paths(strokeWidth:=0.25, stroke="#7f7f7f") %>%

```

```

layer_points(data=county_centers, x=~x, y=~y, size:=8) %>%
layer_text(data=county_centers,
  x=~x+0.05, y=~y, text:=~name,
  baseline:="middle", fontSize:=8) %>%
hide_legend("fill") %>%
hide_axis("x") %>% hide_axis("y") %>%
set_options(width=400, height=600, keep_aspect=TRUE) %>%
bind_shiny("maine_labels")

```

Third plot -----

```

me_pop <- read.csv("data/me_pop.csv", stringsAsFactors=FALSE)
me_crime <- read.csv("data/me_crime.csv", stringsAsFactors=FALSE)

crime_1k <- me_crime %>%
  filter(year==2013) %>%
  select(1,5:12) %>%
  left_join(me_pop) %>%
  mutate(murder_1k=1000*(murder/population_2010),
    rape_1k=1000*(rape/population_2010),
    robbery_1k=1000*(robbery/population_2010),
    aggravated_assault_1k=1000*(aggravated_assault/population_2010),
    burglary_1k=1000*(burglary/population_2010),
    larceny_1k=1000*(larceny/population_2010),
    motor_vehicle_theft_1k=1000*(motor_vehicle_theft/population_2010),
    arson_1k=1000*(arson/population_2010))

map %<>% mutate(id=gsub(" County, ME", "", id)) %>%
  left_join(crime_1k, by=c("id"="county"))

crime_values <- function(x) {
  if(is.null(x)) return(NULL)
  y <- me_crime %>% filter(year==2013, county==x$id) %>% select(1,5:12)
  sprintf("<table width='100%'>%s</table>",
    paste0("<tr><td style='text-align:left>", names(y),
      ":",</td><td style='text-align:right>", format(y),
      collapse="</td></tr>"))
}

map %>%
  group_by(group, id) %>%
  ggvis(~long, ~lat) %>%
  layer_paths(fill=input_select(label="Crime:",
    choices=crime_1k %>%
      select(ends_with("1k")) %>%
      colnames %>% sort,
    id="Crime1",
    map=as.name),
    strokeWidth:=0.5, stroke="white") %>%
  scale_numeric("fill", range=c("#bfd3e6", "#8c6bb1", "#4d004b")) %>%
  add_tooltip(crime_values, "hover") %>%
  add_legend("fill", title="Crime Rate/1K Pop") %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=400, height=600, keep_aspect=TRUE) %>%
  bind_shiny("maine_crime_1", "maine_crime_1_ui")

```

Fourth plot -----

```

map %>%
  mutate_each(funs(log1p), ends_with("1k")) %>%
  group_by(group, id) %>%
  ggvis(~long, ~lat) %>%
  layer_paths(fill=input_select(label="Crime:",
    choices=crime_1k %>%
      select(ends_with("1k")) %>%
      colnames %>% sort,
    id="Crime2",
    map=as.name),
    strokeWidth:=0.5, stroke="black") %>%
  scale_numeric("fill", trans="quantile",
    domain=c(0, crime_1k %>% mutate_each(funs(log1p),
      ends_with("1k")) %>%
      select(ends_with("1k")) %>% max),
    range=brewer.pal(9, "BuPu")) %>%
  add_tooltip(crime_values, "hover") %>%
  add_legend("fill", title="Crime Rate/1K Pop") %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=400, height=600, keep_aspect=TRUE)%>%

```

```

bind_shiny("maine_crime_2", "maine_crime_2_ui")

# Fifth plot -----

#us <- readOGR("data/us.geojson", "OGRGeoJSON")
### EXECUTAR NO WINDOWS
us <- readOGR("data/us.geojson")
us <- us[!us$STATEFP %in% c("02", "15", "72"),]

us_aea <- spTransform(us, CRS("+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b=6370997 +units=m +no_defs"))

map <- ggplot2::fortify(us_aea, region="GEOID")

droughts <- read.csv("data/dm_export_county_20141223.csv")
droughts$id <- sprintf("%05d", as.numeric(as.character(droughts$FIPS)))
droughts$total <- with(droughts, (D0+D1+D2+D3+D4)/5)

map_d <- merge(map, droughts, all.x=TRUE)

ramp <- colorRampPalette(c("white", brewer.pal(n=9, name="YlOrRd")), space="Lab")

map_d$fill_col <- as.character(cut(map_d$total, seq(0,100,10), include.lowest=TRUE, labels=ramp(10)))
map_d$fill_col <- ifelse(is.na(map_d$fill_col), "FFFFFF", map_d$fill_col)

drought_values <- function(x) {
  if(is.null(x) | !(x$id %in% droughts$id)) return(NULL)
  y <- droughts %>% filter(id==x$id) %>% select(1,3,4,6:10)
  sprintf("<table width='100%'>%s</table>",
    paste0("<tr><td style='text-align:left>", names(y),
      ":",</td><td style='text-align:right>", format(y), collapse="</td></tr>"))
}

map_d %>%
  group_by(group, id) %>%
  ggvis(~long, ~lat) %>%
  layer_paths(fill=~fill_col, strokeOpacity := 0.5, strokeWidth := 0.25) %>%
  add_tooltip(drought_values, "hover") %>%
  hide_legend("fill") %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=900, height=600, keep_aspect=TRUE) %>%
  bind_shiny("drought")

# Sixth plot -----

#world <- readOGR("data/ne_50m_admin_0_countries.geojson", layer="OGRGeoJSON")
### EXECUTAR NO WINDOWS
world <- readOGR("data/ne_50m_admin_0_countries.geojson")
world <- world[!world$iso_a3 %in% c("ATA"),]
world <- spTransform(world, CRS("+proj=wintri"))

map_w <- ggplot2::fortify(world, region="iso_a3")

launch_sites <- rbindlist(lapply(ogrListLayers("data/launch-sites.kml"), function(layer) {
  tmp <- readOGR("data/launch-sites.kml", layer)
  places <- data.table(coordinates(tmp)[,1:2], as.character(tmp$Name))
})))

setnames(launch_sites, colnames(launch_sites), c("lon", "lat", "name"))
coordinates(launch_sites) <- ~lon+lat
launch_sites <- as.data.frame(SpatialPointsDataFrame(spTransform(
  SpatialPoints(launch_sites, CRS("+proj=longlat")), CRS("+proj=wintri")),
  launch_sites@data))

map_w %>%
  group_by(group, id) %>%
  ggvis(~long, ~lat) %>%
  layer_paths(fill="#252525", stroke="white", strokeOpacity:=0.5, strokeWidth:=0.25) %>%
  layer_points(data=launch_sites, stroke="white", x=~lon, y=~lat, fill="#cb181d", size=25, fillOpacity:=0.5, strokeWidth:=0.25) %>%
  hide_legend("fill") %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=900, height=500, keep_aspect=TRUE) %>%
  bind_shiny("launch")

# Turn off progress bar -----

progress$close()

})

```

```

# This is the user-interface definition of a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)
library(ggvis)

me_crime <- read.csv("data/me_crime.csv", stringsAsFactors=FALSE)

shinyUI(fluidPage(
  titlePanel("ggvis shiny maps"),
  sidebarLayout(
    sidebarPanel(
      p(strong("NOTE: "), "Since this loads many maps, it can take a few seconds to startup. Even after that, there may be some delay with more complex maps due to how ggvis renders them."),
      hr(),
      p("This is an example of making static & interactive maps with ggvis and wiring them up interactively in a Shiny app."),
      p("The first two render static Maine state maps (with & without county labels). The second two let you interactively explore Maine county crime data for 2013 (by 1K population)."),
      p("The U.S. Drought Map interactively shows drought levels as of 2014-12-23 and is also an example of using a projection (Albers) & also custom colors outside of any ggvis scale). It also shows that many polygons can take a while to render (initially)."),
      p("The final example is a world map with a Winkel-Tripel projection and also shows how to add projected points to the map (no interactivity)."),
      hr(),
      p("Written by ", a(href="http://twitter.com/hrbrmstr", "@hrbrmstr")),
      p("Source on ", a(href="https://github.com/hrbrmstr/ggvis-maps", "github")),
      p("Static version on ", a(href="http://rpubs.com/hrbrmstr/ggvis-maps", "RPubs")),
      width=3
    ),
    mainPanel(
      tabsetPanel(
        tabPanel("Maine", wellPanel(p("This is just a basic map outline. Since ggvis has no ", code("coord_map")), " we have to get creative with a fixed aspect ratio and good choices for height & width.")), ggvisOutput("maine")),
        tabPanel("Maine (Labels)", wellPanel(p("Again, no interaction here, but this example shows how to annotate a static map with points and labels.")), ggvisOutput("maine_labels")),
        tabPanel("Maine Crime", wellPanel(p("We finally get some interaction and also add tooltips. We use ", code("input_select"), " to enable exploration between different crimes.")), uiOutput("maine_crime_1_ui"), ggvisOutput("maine_crime_1")),
        tabPanel("Maine Crime (log)", wellPanel(p("This is the same as the previous one, but uses the full range of all crimes for the fill color (log scale).")), uiOutput("maine_crime_2_ui"), ggvisOutput("maine_crime_2")),
        tabPanel("US Drought", wellPanel(p("This shows how to use a projection and also a custom color scale without using any of the built-in ", code("scale_"), " functions. It also shows the use of a tooltip where there is no data for some polygons.")), ggvisOutput("drought")),
        tabPanel("Global Launch Sites", wellPanel(p("The last example shows how to both use a projection and also how to project other points to display on the map.")), ggvisOutput("launch"))
      ),
      width=9
    )
  )
))

```

Criando Mapas com RGoogleMaps

```

# Mapas

# Google maps
# https://www.rdocumentation.org/packages/RgoogleMaps/versions/1.4.1
# https://cran.r-project.org/web/packages/RgoogleMaps/RgoogleMaps.pdf

# RGDal
# https://cran.r-project.org/web/packages/rgdal/index.html

# Diretório de trabalho
setwd("~/Dropbox/DSA/Data Viz/Cap06/06-Mapas")

# Pacotes
install.packages("rgdal")
install.packages("RgoogleMaps")
library(rgdal)
library(RgoogleMaps)

# Dataset
air <- read.csv("londonair.csv")
View(air)

```

```

# Gravando o mapa 1
london <- GetMap(center = c(51.51,-0.116), zoom = 10, destfile = "London.png", maptype = "mobile")

# Gerando o mapa 1
PlotOnStaticMap(london,lat = air$lat, lon = air$lon, cex = 2, pch = 19, col = as.character(air$color))

# Gravando o mapa 2
london <- GetMap(center = c(51.51,-0.116), zoom = 10, destfile = "London_satellite.png", maptype = "satellite")

# Gerando o mapa 2
PlotOnStaticMap(london,lat = air$lat, lon = air$lon, cex = 2, pch = 19, col = as.character(air$color))

# Obter os dados do mapa
GetMap(center = c(40.714728,-73.99867), zoom = 14, destfile = "Manhattan.png", maptype = "hybrid")

# Gerando o gráfico a partir das variáveis e alterando o tipo
library(RgoogleMaps)

# ylim
lat <- c(48,64)

# xlim
lon <- c(-140,-110)

# Como centralizar o mapa
center = c(mean(lat), mean(lon))

# zoom
zoom <- 5

# Diversas opções visuais do google maps:
# maptype = c("roadmap", "mobile", "satellite", "terrain", "hybrid", "mapmaker-roadmap", "mapmaker-hybrid")
terrrmap <- GetMap(center = center, zoom = zoom, maptype = "terrain", destfile = "terrain.png")


# Toronto Traffic Signals Heat Map
# Myles Harrison
# http://www.everydayanalytics.ca
# Data from Toronto Open Data Portal:
# http://www.toronto.ca/open

library(MASS)
library(RgoogleMaps)
library(RColorBrewer)

# addalpha() define as cores que serão usadas no mapa
addalpha <- function(colors, alpha = 1.0) {
  r <- col2rgb(colors, alpha=T)
  r[4,] <- alpha*255
  r <- r/255.0
  return(rgb(r[1,], r[2,], r[3,], r[4,]))
}

# colorRampPaletteAlpha() cria a paleta de cores
colorRampPaletteAlpha <- function(colors, n = 32, interpolate = 'linear') {
  cr <- colorRampPalette(colors, interpolate=interpolate)(n)
  a <- col2rgb(colors, alpha=T)[4,]
  if (interpolate=='linear') {
    l <- approx(a, n=n)
  } else {
    l <- spline(a, n=n)
  }
  l$y[l$y > 255] <- 255
  cr <- addalpha(cr, l$y/255.0)
  return(cr)
}

# Carregando os dados
data <- read.csv(file = "traffic_signals.csv", skip = 1, header = T, stringsAsFactors = F)
View(data)

# Coletando dados de latitude e longitude
rawdata <- data.frame(as.numeric(data$Longitude), as.numeric(data$Latitude))
names(rawdata) <- c("lon", "lat")

```

```

data <- as.matrix(rawdata)

# Rodar as coordenadas lat-lon usando uma matriz de rotação
# Teste e erro conduzem a  $\pi / 15.0 = 12$  graus
theta = pi/15.0
m = matrix(c(cos(theta), sin(theta), -sin(theta), cos(theta)), nrow=2)
data <- as.matrix(data) %*% m

# Parâmetros e Plot
par(bg = 'black')
plot(data, cex = 0.1, col = "white", pch = 16)

# Crie o heatmap com kde2d e overplot
k <- kde2d(data[,1], data[,2], n=500)

# Intensidade de verde para vermelho
cols <- rev(colorRampPalette(brewer.pal(8, 'RdYlGn'))(100))
par(bg = 'white')
image(k, col = cols, xaxt = 'n', yaxt = 'n')
points(data, cex = 0.1, pch = 16)

# Mapeamento via RgoogleMaps
# Localizar centro do mapa e obter mapa
center <- rev(sapply(rawdata, mean))
map <- GetMap(center = center, zoom = 11)

# Traduzir dados originais
coords <- LatLon2XY.centered(map, rawdata$lat, rawdata$lon, 11)
coords <- data.frame(coords)

# Gera o Heat Map novamente
k2 <- kde2d(coords$newX, coords$newY, n = 500)

# Crie um vetor de transparência exponencial e adicione
alpha <- seq.int(0.5, 0.95, length.out=100)
alpha <- exp(alpha^6-1)
cols2 <- addalpha(cols, alpha)

# Plot
PlotOnStaticMap(map)
image(k2, col = cols2, add = T)
points(coords$newX, coords$newY, pch = 16, cex = 0.3)

```

Gráficos Interativos de Séries Temporais com Dygraphs

```

# Gráficos Interativos de Séries Temporais com Dygraphs

# https://rstudio.github.io/dygraphs/

# Pacote
install.packages("dygraphs")
library(dygraphs)

# Dataset
lungDeaths <- cbind(mdeaths, fdeaths)

# Gráfico
dygraph(lungDeaths)

# Gráfico com seletor na parte inferior
dygraph(lungDeaths) %>% dyRangeSelector()

# Labels
dygraph(lungDeaths) %>%
  dySeries("mdeaths", label = "Male") %>%
  dySeries("fdeaths", label = "Female") %>%
  dyOptions(stackedGraph = TRUE) %>%
  dyRangeSelector(height = 20)

# Gerando previsões
hw <- HoltWinters(ldeaths)
predicted <- predict(hw, n.ahead = 72, prediction.interval = TRUE)

# Plot das previsões

```

```
dygraph(predicted, main = "Predicted Lung Deaths (UK)") %>%
  dyAxis("x", drawGrid = FALSE) %>%
  dySeries(c("lwr", "fit", "upr"), label = "Deaths") %>%
  dyOptions(colors = RColorBrewer::brewer.pal(3, "Set1"))
```

Criando Aplicações Web Interativas com Shiny Dashboard

```
# Google Analytics Dashboard com Shiny
# server.R é o código para conexão e tratamento dos dados

# Pacotes
# Instalar os pacotes no console do RStudio, caso não estejam instalados no seu computador
# Para instalar os pacotes, digite: install.packages("nome_pacote")
library(shiny)
library(dplyr)
library(ggplot2)
library(rgdal)
library(RColorBrewer)
library(googleVis)
library(leaflet)

shinyServer(function(input, output, session) {

  # Carrega os dados
  load("gadf.Rdata")

  # Template da função abaixo em: http://shiny.rstudio.com/articles/client-data.html
  cdata <- session$clientData

  # Valores retornados como texto
  output$clientdataText <- renderText({
    cnames <- names(cdata)
    allvalues <- lapply(cnames, function(name) {
      paste(name, cdata[[name]], sep = " = ")
    })
    paste(allvalues, collapse = "\n")
  })

  # Filtrando os dados
  passData <- reactive({
    firstData <- filter(gadf, date >= input$dateRange[1] & date <= input$dateRange[2])
    if(!is.null(input$domainShow)){
      firstData <- filter(firstData, networkDomain %in% input$domainShow)
    }
    return(firstData)
  })

  # Ícones
  output$days <- renderInfoBox({
    infoBox(
      "Dias", input$dateRange[2] - input$dateRange[1],
      icon = icon("calendar", lib = "font-awesome"),
      color = "blue",
      fill = ifelse(max(passData())$date) == max(gadf$date),
        TRUE, FALSE)
  })

  output$users <- renderInfoBox({
    infoBox(
      "Usuários", sum(passData())$users, icon = icon("user"),
      color = "purple",
      fill = ifelse(sum(passData())$users /
        as.numeric(input$dateRange[2] -
          input$dateRange[1]) > 20,
        TRUE, FALSE)
  })

  output$percentNew <- renderInfoBox({
    infoBox(
      "Novos Usuários",
```



```

paste0(round(sum(passData())$newUsers) /
        sum(passData())$users * 100, 1), "%"),
icon = icon("pie-chart"),
color = "green",
fill = ifelse(sum(passData())$newUsers) /
        sum(passData())$users * 100 > 50,
        TRUE, FALSE)
)
})

output$notifications <- renderMenu({

  users <- sum(gadf[gadf$date == max(gadf$date), "users"])
  newusers <- sum(gadf[gadf$date == max(gadf$date), "newUsers"]) / sum(gadf[gadf$date == max(gadf$date), "users"]) * 100
  newusers <- round(newusers, 0)
  notifData <- data.frame("number" = c(users, newusers),
    "text" = c(" users today", "% new users"),
    "icon" = c("users", "user"))

  notifs <- apply(notifData, 1, function(row) {
    notificationItem(text = paste0(row[["number"]], row[["text"]]),
      icon = icon(row[["icon"]]))
  })

  dropdownMenu(type = "notifications", .list = notifs)

})

output$gauge <- renderGvis({
  session$clientData$output_trend_width

  df <- data.frame(Label = "Bounce %", Value = round(mean(passData())$bounceRate, trim = .1), 1))

  gvisGauge(df,
    options = list(min = 0, max = 100, greenFrom = 0,
      greenTo = 50, yellowFrom = 50, yellowTo = 70,
      redFrom = 70, redTo = 100))

})

output$trend <- renderPlot({

  groupByDate <- group_by(passData(), YearMonth, networkDomain) %>%
    summarise(meanSession = mean(sessionDuration),
      users = sum(users),
      newUsers = sum(newUsers), sessions = sum(sessions))

  groupByDate$Date <- as.Date(paste0(groupByDate$YearMonth, "01"), format = "%Y%m%d")

  thePlot <- ggplot(groupByDate,
    aes_string(x = "Date", y = input$outputRequired,
      group = "networkDomain", colour = "networkDomain")) +
    geom_line()

  if(input$smooth){
    thePlot <- thePlot + geom_smooth()
  }

  print(thePlot)

})

output$histogram <- renderPlot({

  groupByDate <- group_by(passData(), YearMonth, networkDomain) %>%
    summarise(meanSession = mean(sessionDuration),
      users = sum(users),
      newUsers = sum(newUsers), sessions = sum(sessions))

  groupByDate$Date <- as.Date(paste0(groupByDate$YearMonth, "01"), format = "%Y%m%d")

  ggplot(groupByDate,
    aes_string(x = input$outputRequired, group = "networkDomain")) +
    geom_histogram(binwidth = diff(range(groupByDate[[input$outputRequired]])) / 10)

})

```

```

# Produzindo o Mapa
output$ggplotMap <- renderPlot ({
  groupCountry <- group_by(passData(), country)
  groupByCountry <- summarise(groupCountry, meanSession = mean(sessionDuration),
    users = log(sum(users)), sessions = log(sum(sessions)))

  world <- readOGR(dsn = ".", layer = "world_country_admin_boundary_shapefile_with_fips_codes")
  countries <- world@data
  countries <- cbind(id = rownames(countries), countries)
  countries <- merge(countries, groupByCountry, by.x = "CNTRY_NAME", by.y = "country", all.x = TRUE)
  map.df <- fortify(world)
  map.df <- merge(map.df, countries, by = "id")

  ggplot(map.df, aes(x = long, y = lat, group = group)) +
    geom_polygon(aes_string(fill = input$outputRequired)) +
    geom_path(colour = "grey50") +
    scale_fill_gradientn(colours = rev(brewer.pal(9, "Spectral")),
      na.value = "white") +
    coord_fixed() + labs(x = "", y = "")

})

# Leaflet map
output$leaflet <- renderLeaflet({
  leaflet(gadf) %>% addTiles() %>% setView(lng = 1.1333, lat = 52.95, zoom = 4)
})

})

```

```

# Google Analytics Dashboard com Shiny
# ui.R é o código para interface com o usuário

# Pacotes
# Instalar os pacotes no console do RStudio, caso não estejam instalados no seu computador
# Para instalar os pacotes, digite: install.packages("nome_pacote")
library(shiny)
library(shinydashboard)
library(shinyBS)
library(leaflet)

header <- dashboardHeader(title = "Google Analytics", dropdownMenuOutput("notifications"))

sidebar <- dashboardSidebar(
  sidebarMenu(
    menuItem("Dashboard", tabName = "dashboard",
      icon = icon("dashboard")),

    menuItem("Mapas", icon = icon("globe"), tabName = "map",
      badgeColor = "red"),

    dateRangeInput(inputId = "dateRange", label = "Período de Data",
      start = "2013-05-01"),

    radioButtons(inputId = "outputRequired",
      label = "Selecione uma Opção",
      choices = list("Tempo Médio de Sessão" = "meanSession",
        "Usuários" = "users",
        "Sessões" = "sessions")),

    checkboxInput("smooth", label = "Adicionar Margem de Erro?",
      value = FALSE),

    actionButton("showData", "Mostrar os Dados de Conexão")
  )
)

body <- dashboardBody(
  bsModal(id = "clientData", title = "Client Data",
    trigger = "showData",
    verbatimTextOutput("clientdataText")),
  tabItems(
    tabItem(tabName = "dashboard",
      fluidRow(
        infoBoxOutput(width = 3, "days"),
        infoBoxOutput(width = 3, "users"),
        infoBoxOutput(width = 3, "percentNew"),
        infoBox(width = 3, "Versão do Shiny", "0.12",
          icon = icon("desktop")))
    )
  )
)

```

```
fluidRow(
  box(width = 5, plotOutput("trend")),
  box(width = 2, htmlOutput("gauge")),
  box(width = 5, plotOutput("histogram"))
),
tabItem(tabName = "map",
  box(width = 6, plotOutput("ggplotMap")),
  box(width = 6, leafletOutput("leaflet")))
)
dashboardPage(header, sidebar, body)
```

Quizz

O gráfico é uma representação com forma geométrica construída de maneira exata e precisa a partir de informações numéricas obtidas através de pesquisas ou organizadas em uma tabela.

O R traz em sua instalação padrão, um pacote básico de plotagem, chamado de Base Plotting System.

A gramática de gráficos é usada para descrever as características que fundamentam todos os gráficos, principalmente os gráficos estatísticos.

No ggplot2, os gráficos são construídos camada por camada. Cada camada representa um tipo de mapeamento ou personalização do gráfico.

Os geoms definem qual forma geométrica será utilizada para a visualização dos dados no gráfico criado com ggplot2.

9.7. Visualização de Dados com D3.js

Criando Elementos com D3.js

???

O Que É SVG?

Uma das grandes tendências do momento quando se fala sobre desenvolvimento web é o formato SVG, principalmente com o advento do design responsivo e a consequente preocupação com dispositivos com densidade de pixel superior (HiDPI) como a tela retina da Apple, utilizada nos modelos mais recentes do iPhone, iPad e do Macbook Pro. Mas o que exatamente é um arquivo SVG? Qual é a diferença entre um vetor e um bitmap? E como e por que utilizar esta tecnologia a nosso favor? Formato SVG combinados com D3.js formam uma poderosa ferramenta de visualização de dados.

Criado pelo W3C o SVG (Scalable Vectorial Graphics), que em português significa Vetor Gráfico Redimensionável, é nada mais que um arquivo XML que contem tags específicas para gerar uma imagem vetorizada na sua aplicação. Com tags bastante simples você consegue gerar imagens de alta qualidade vetorizadas que por mais que você altere as proporções na tela essa não perderá qualidade, por ser uma imagem vetorizada.

Podendo ser apenas imagem fixa ou animação, o SVG pode ser trabalhado junto ao JavaScript para manipular eventos de imagem. O formato SVG permite três tipos de objetos gráficos, sendo eles imagens, textos ou formas geométricas vetoriais. Um arquivo SVG é basicamente um mapa em XML que descreve matematicamente uma figura gráfica bidimensional. Funciona como um conjunto de instruções numéricas para realizar um desenho, que são convertidas em imagens em um software capaz de interpreta-lo (como um browser, por exemplo). SVG é para uma imagem, o que o HTML é para um texto.

Por que utilizar SVG?

Quando você trabalha com imagens que necessitam de grande riqueza de detalhes e vai exibir esta em uma aplicação, essa imagem precisa ser salva com uma grande qualidade, ou seja, essa imagem vai ter um grande tamanho e custo de armazenamento. Caso se queira economizar espaço a imagem vai ser gerada com menos qualidade e ao expandir essa imagem irá distorcer, espalhando os pixels e deixando visível isso.

Como no SVG os dados são escaláveis a imagem pode ser redimensionada sem preocupação quanto a qualidade e distorção dessa imagem, sem falar que por ser apenas um arquivo XML, apenas texto, o custo de armazenamento é muito inferior assim como o de exibição. Na internet, por exemplo, se a imagem for grande dependendo da conexão do usuário essa vai levar um certo tempo para ser carregada e utilizando o svg a imagem é renderizada mais rapidamente, pois o browser lê as tags do xml e vai construindo a imagem na página. Muitas aplicações mobile hoje já estão utilizando o SVG, pois como a maioria dos aplicativos não possuem um grande potencial de hardware, é menos custoso exibir imagens e armazenar com essa extensão.

Vetor vs Bitmap

Existem dois tipos principais de arquivos de imagens. Vetores e Bitmaps. Os arquivos do tipo vetor (como AI, EPS, CDR e o nosso novo melhor amigo SVG) são linhas, curvas e formas geométricas descritas matematicamente. Já os arquivos bitmaps (como JPG, PNG, GIF etc) são compostos por um grid de pixels.

As vantagens de se utilizar gráficos em vetor são incríveis. Primeiramente por que é possível redimensiona-los infinitamente sem perder qualidade ou nitidez. Na prática isto significa que um ícone, por exemplo, terá a mesma aparência sem distorções em um smartphone ou uma televisão de 42". Não importa qual é a quantidade de espaço que a imagem ocupa, o arquivo terá o mesmo peso. O que potencialmente economiza a quantidade de banda necessária para realizar o download, minimiza a quantidade de requisições para o servidor já que não são necessárias diversas imagens diferentes para servir todas as necessidades, etc... A possibilidade de uso de medidas relativas também faz dos gráficos em SVG o formato ideal para se trabalhar com design responsivo.

Estrutura Básica do SVG

Inicialmente se você sabe XML você sabe SVG, se você não sabe XML estude antes de estudar SVG, vai ficar mais muito mais fácil assim. Abaixo a estrutura básica do SVG:

Os parâmetros width e height correspondem a largura e a altura (respectivamente) do arquivo SVG.

O parâmetro viewBox é utilizado para descrever a caixa de visualização da imagem SVG, como se fosse a janela, um iframe que ira montar a imagem dentro.

Arrays e Objetos

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Visualização de Dados com D3</title>
  <link rel="stylesheet" href="main.css">
  <script type="text/javascript" src="d3.min.js"></script>
</head>
<body>

  <!-- Aqui são colocados os elementos DOM -->
  <p>Arrays e Objetos</p>

<script>

// Criando um array de médias de notas de disciplinas de um curso de Ciência da Computação
var medias = [78,71,67,93,59,43,38,89,91,80,67,75];
console.log(medias[0]);

var disciplina = {key: "Algoritmos I", value: 71};
console.log(disciplina.key, disciplina.value);

var disciplinas = [
  {key: "Algoritmos I",          value: 78},
  {key: "Algoritmos II",         value: 71},
  {key: "Programação Java",      value: 67},
  {key: "Programação Python",    value: 93}
];
console.log(disciplinas[1].key, disciplinas[1].value);

// Loop for para percorrer o array e imprimir no console os valores
for(var i = 0, len = disciplinas.length; i < len; i++){
  console.log(disciplinas[i].key, disciplinas[i].value);
}

// Equivalente ao loop for
disciplinas.forEach(function(entry){
  console.log(entry.key, entry.value);
});
```

```
</script>
</body>
</html>
```

Função Filter e Função Map

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Visualização de Dados com D3</title>
  <link rel="stylesheet" href="main.css">
  <script type="text/javascript" src="d3.min.js"></script>
</head>
<body>

  <!-- Aqui são colocados os elementos DOM -->
  <p>Funções Filter e Map</p>

  <script>
var medias = [78,71,67,93,59,43,88,89,91,80,67,75];
var mediaMaior70 = medias.filter(function(entry){
  return entry > 70;
});
//console.log("Médias Maiores que 70: ", mediaMaior70);

var disciplinas = [
  {key: "Lógica", value: 78},
  {key: "Álgebra I", value: 71},
  {key: "Orientação a Objetos", value: 67},
  {key: "Estatística I", value: 93},
  {key: "Sistemas Operacionais I", value: 59},
  {key: "Física Experimental", value: 43},
  {key: "Engenharia de Software", value: 88},
  {key: "Programação Python", value: 89},
  {key: "Algoritmos I", value: 91},
  {key: "Algoritmos II", value: 80},
  {key: "Programação Java", value: 67},
  {key: "Programação Python", value: 75}
];
var mediaMaior70 = disciplinas.filter(function(entry){
  return entry.value > 70;
});
//console.log("Disciplinas com Notas Médias Superiores a 70: ", mediaMaior70);

var reformatDisciplinas = disciplinas.map(function(entry){
  return {
    disciplina: entry.key,
    media: entry.value
  };
});
reformatDisciplinas.forEach(function(entry){
  console.log(entry);
});
</script>
</body>
</html>
```

Funções Especiais do D3.js

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Visualização de Dados com D3</title>
  <link rel="stylesheet" href="main.css">
  <script type="text/javascript" src="d3.min.js"></script>
</head>
<body>

  <!-- Aqui são colocados os elementos DOM -->
```

```
<p>Funções Especiais em D3</p>

<script>

// Dados
var medias = [78,71,67,93,59,43,88,89,91,80,67,75];

// Funções
var menorMedia = d3.min(medias);
var maiorMedia = d3.max(medias);
var mediaMedias = d3.mean(medias);
//console.log(menorMedia, maiorMedia, mediaMedias);

// Funções com Arrays
var disciplinas = [
  {key: "Lógica", value: 78},
  {key: "Álgebra I", value: 71},
  {key: "Orientação a Objetos", value: 67},
  {key: "Estatística I", value: 93},
  {key: "Sistemas Operacionais I", value: 59},
  {key: "Física Experimental", value: 43},
  {key: "Engenharia de Software", value: 88},
  {key: "Programação Python", value: 89},
  {key: "Algoritmos I", value: 91},
  {key: "Algoritmos II", value: 80},
  {key: "Programação Java", value: 67},
  {key: "Programação Python", value: 75}
];

var disciplinasMenorMedia = d3.min(disciplinas, function(d){
  return d.value;
});
var disciplinasMaiorMedia = d3.max(disciplinas, function(d){
  return d.value;
});
var disciplinasMediaMedias = d3.mean(disciplinas, function(d){
  return d.value;
});
console.log(disciplinasMenorMedia, disciplinasMaiorMedia, disciplinasMediaMedias);
</script>
</body>
</html>
```

Chart com SVG

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Visualização de Dados com D3</title>
  <link rel="stylesheet" href="main.css">
  <script type="text/javascript" src="d3.min.js"></script>
</head>
<body>

<!-- Aqui são colocados os elementos DOM -->
<!-- <svg id="chart"></svg>-->

<script>

var disciplinas = [
  {key: "Lógica", value: 78},
  {key: "Álgebra I", value: 71},
  {key: "Orientação a Objetos", value: 67},
  {key: "Estatística I", value: 93},
  {key: "Sistemas Operacionais I", value: 59},
  {key: "Física Experimental", value: 43},
  {key: "Engenharia de Software", value: 88},
  {key: "Estrutura de Dados", value: 36},
  {key: "Algoritmos I", value: 91},
  {key: "Algoritmos II", value: 74},
  {key: "Programação Java", value: 67},
  {key: "Programação Python", value: 75}
];
```

```

// Variáveis de largura e altura da área de desenho do SVG
var w = 800;
var h = 450;

// Margens
var margin = {
    top: 58,
    bottom: 150,
    left: 80,
    right: 40
};

// Definindo a área em que o gráfico será construído dentro da área de desenho
var width = w - margin.left - margin.right;
var height = h - margin.top - margin.bottom;

// Definindo x com escala ordinal
var x = d3.scale.ordinal()
    .domain(disciplinas.map(function(entry){
        return entry.key;
    }))
    .rangeBands([0, width]);

// Definindo y com escala linear
var y = d3.scale.linear()
    .domain([0, d3.max(disciplinas, function(d){
        return d.value;
    })])
    .range([height, 0]);

// Definindo as cores das barras
var ordinalColorScale = d3.scale.category20();

// Desenhando o eixo x
var xAxis = d3.svg.axis()
    .scale(x)
    .orient("bottom");

// Desenhando o eixo y
var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left");

// Definindo linhas de grid
var yGridlines = d3.svg.axis()
    .scale(y)
    .tickSize(-width,0,0)
    .tickFormat("")
    .orient("left");

// Definindo o objeto SVG
var svg = d3.select("body").append("svg")
    .attr("id", "chart")
    .attr("width", w)
    .attr("height", h);

// Definindo um objeto SVG agrupado
var chart = svg.append("g")
    .classed("display", true)
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

// Adicionando função de controle
var controls = d3.select("body")
    .append("div")
    .attr("id", "controls");

// Adicionando um botão de ação
var sort_btn = controls.append("button")
    .html("Ordenar os Dados: ascending")
    .attr("state", 0);

// Função para desenhar os eixos
function drawAxis(params){
    if(params.initialize === true){
        // Desenha as gridlines
        this.append("g")
            .call(params.gridlines)
    }
}

```



```

        .classed("gridline", true)
        .attr("transform", "translate(0,0)")

// Eixo x
this.append("g")
    .classed("x axis", true)
    .attr("transform", "translate(" + 0 + "," + height + ")")
    .call(params.axis.x)
        .selectAll("text")
            .classed("x-axis-label", true)
            .style("text-anchor", "end")
            .attr("dx", -8)
            .attr("dy", 8)
            .attr("transform", "translate(0,0) rotate(-45)");

// Eixo y
this.append("g")
    .classed("y axis", true)
    .attr("transform", "translate(0,0)")
    .call(params.axis.y);

// Label y
this.select(".y.axis")
    .append("text")
    .attr("x", 30)
    .attr("y", 0)
    .style("text-anchor", "middle")
    .attr("transform", "translate(-50," + height/2 + ") rotate(-90)")
    .text("Médias");

// Label x
this.select(".x.axis")
    .append("text")
    .attr("x", 0)
    .attr("y", 0)
    .style("text-anchor", "middle")
    .attr("transform", "translate(" + width/2 + ",145)")
    .text("Disciplinas");

} else if(params.initialize === false){
    // Código alternativo caso as variáveis não seja inicializadas
    this.selectAll("g.x.axis")
        .transition()
        .duration(500)
        .ease("bounce")
        .delay(500)
        .call(params.axis.x);
    this.selectAll(".x-axis-label")
        .style("text-anchor", "end")
        .attr("dx", -8)
        .attr("dy", 8)
        .attr("transform", "translate(0,0) rotate(-45)");
    this.selectAll("g.y.axis")
        .transition()
        .duration(500)
        .ease("bounce")
        .delay(500)
        .call(params.axis.y);
}

}

// Função que desenha o gráfico
function plot(params){

    // Obtém a coluna key do nosso conjunto de dados
    x.domain(disciplinas.map(function(entry){
        return entry.key;
    }));

    // Obtém a coluna value do nosso conjunto de dados
    y.domain([0, d3.max(disciplinas, function(d){
        return d.value;
    })]);

    // Desenha os labels nos eixos x e y
    drawAxis.call(this, params);

    // Interatividade ao passar o mouse pelas barras

```

```

        this.selectAll(".bar")
            .data(params.data)
            .enter()
                .append("rect")
                .classed("bar", true)
                .on("mouseover", function(d,i){
                    d3.select(this).style("fill", "yellow");
                })
                .on("mousemove", function(d,i){

                })
                .on("mouseout", function(d,i){
                    d3.select(this).style("fill", ordinalColorScale(i));
                });

        this.selectAll(".bar-label")
            .data(params.data)
            .enter()
                .append("text")
                .classed("bar-label", true);

// Define o gráfico de barras
this.selectAll(".bar")
    .transition()
    .attr("x", function(d,i){
        return x(d.key);
    })
    .attr("y", function(d,i){
        return y(d.value);
    })
    .attr("height", function(d,i){
        return height - y(d.value);
    })
    .attr("width", function(d){
        return x.rangeBand();
    })
    .style("fill", function(d,i){
        return ordinalColorScale(i);
    });

// Define os labels
this.selectAll(".bar-label")
    .transition()
    .attr("x", function(d,i){
        return x(d.key) + (x.rangeBand()/2)
    })
    .attr("dx", 0)
    .attr("y", function(d,i){
        return y(d.value);
    })
    .attr("dy", -6)
    .text(function(d){
        return d.value;
    })

this.selectAll(".bar")
    .data(params.data)
    .exit()
    .remove();

this.selectAll(".bar-label")
    .data(params.data)
    .exit()
    .remove();
}

// Ordena as barras ao clicar no botão
sort_btn.on("click", function(){
    var self = d3.select(this);
    var ascending = function(a,b){
        return a.value - b.value;
    };
    var descending = function(a,b){
        return b.value - a.value;
    }
    var state = +self.attr("state");
    var txt = "Ordenar os Dados: ";
    if(state === 0){

```

```

        disciplinas.sort(ascending);
        state = 1;
        txt += "descending";
    } else if(state === 1){
        disciplinas.sort(descending);
        state = 0;
        txt += "ascending";
    }
    self.attr("state", state);
    self.html(txt);

    plot.call(chart, {
        data: disciplinas,
        axis:{
            x: xAxis,
            y: yAxis
        },
        gridlines: yGridlines,
        initialize: false
    });
});

// Chamada à função
plot.call(chart, {
    data: disciplinas,
    axis:{
        x: xAxis,
        y: yAxis
    },
    gridlines: yGridlines,
    initialize: true
});

</script>
</body>
</html>

```

Criando um Plot de Series Temporais

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Visualização de Dados com D3</title>
    <link rel="stylesheet" href="main.css">
    <script type="text/javascript" src="d3.min.js"></script>
</head>
<body>

    <!-- Aqui são colocados os elementos DOM -->
    <!-- <svg id="chart"></svg>-->

<script>

var data = [
    {key: "Petrobras", value: 60, date: "2017/01/01" },
    {key: "Petrobras", value: 58, date: "2017/01/02" },
    {key: "Petrobras", value: 59, date: "2017/01/03" },
    {key: "Petrobras", value: 56, date: "2017/01/04" },
    {key: "Petrobras", value: 57, date: "2017/01/05" },
    {key: "Petrobras", value: 55, date: "2017/01/06" },
    {key: "Petrobras", value: 56, date: "2017/01/07" },
    {key: "Petrobras", value: 52, date: "2017/01/08" },
    {key: "Petrobras", value: 54, date: "2017/01/09" },
    {key: "Petrobras", value: 57, date: "2017/01/10" },
    {key: "Petrobras", value: 56, date: "2017/01/11" },
    {key: "Petrobras", value: 59, date: "2017/01/12" },
    {key: "Petrobras", value: 56, date: "2017/01/13" },
    {key: "Petrobras", value: 52, date: "2017/01/14" },
    {key: "Petrobras", value: 48, date: "2017/01/15" },
    {key: "Petrobras", value: 47, date: "2017/01/16" },
    {key: "Petrobras", value: 48, date: "2017/01/17" },
    {key: "Petrobras", value: 45, date: "2017/01/18" },
    {key: "Petrobras", value: 43, date: "2017/01/19" },

```

```

    {key: "Petrobras", value: 41, date: "2017/01/20" },
    {key: "Petrobras", value: 37, date: "2017/01/21" },
    {key: "Petrobras", value: 36, date: "2017/01/22" },
    {key: "Petrobras", value: 39, date: "2017/01/23" },
    {key: "Petrobras", value: 41, date: "2017/01/24" },
    {key: "Petrobras", value: 42, date: "2017/01/25" },
    {key: "Petrobras", value: 40, date: "2017/01/26" },
    {key: "Petrobras", value: 43, date: "2017/01/27" },
    {key: "Petrobras", value: 41, date: "2017/01/28" },
    {key: "Petrobras", value: 39, date: "2017/01/29" },
    {key: "Petrobras", value: 40, date: "2017/01/30" },
    {key: "Petrobras", value: 39, date: "2017/01/31" }
];

var w = 800;
var h = 450;

var margin = {
    top: 58,
    bottom: 100,
    left: 80,
    right: 40
};

var width = w - margin.left - margin.right;
var height = h - margin.top - margin.bottom;

var svg = d3.select("body").append("svg")
    .attr("id", "chart")
    .attr("width", w)
    .attr("height", h);

var chart = svg.append("g")
    .classed("display", true)
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

var dateParser = d3.time.format("%Y/%m/%d").parse;

var x = d3.time.scale()
    .domain(d3.extent(data, function(d){
        var date = dateParser(d.date);
        return date;
    }))
    .range([0,width]);

var y = d3.scale.linear()
    .domain([0, d3.max(data, function(d){
        return d.value;
    })])
    .range([height,0]);

var xAxis = d3.svg.axis()
    .scale(x)
    .orient("bottom")
    .ticks(d3.time.days, 7)
    .tickFormat(d3.time.format("%m/%d"));

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left")
    .ticks(5);

var line = d3.svg.line()
    .x(function(d){
        var date = dateParser(d.date);
        return x(date);
    })
    .y(function(d){
        return y(d.value);
    })
    .interpolate("monotone");

var area = d3.svg.area()
    .x(function(d){
        var date = dateParser(d.date);
        return x(date);
    })
    .y0(height)

```

```

        .y1(function(d){
            return y(d.value);
        })
        .interpolate("monotone");

function plot(params){
    this.append("g")
        .classed("x axis", true)
        .attr("transform", "translate(0," + height + ")")
        .call(params.axis.x);
    this.append("g")
        .classed("y axis", true)
        .attr("transform", "translate(0,0)")
        .call(params.axis.y);
    this.selectAll(".area")
        .data([params.data])
        .enter()
            .append("path")
            .classed("area", true);
    this.selectAll(".trendline")
        .data([params.data])
        .enter()
            .append("path")
            .classed("trendline", true);
    this.selectAll(".point")
        .data(params.data)
        .enter()
            .append("circle")
            .classed("point", true)
            .attr("r", 2);
    this.selectAll(".area")
        .attr("d", function(d){
            return area(d);
        });
    this.selectAll(".trendline")
        .attr("d", function(d){
            return line(d);
        });
    this.selectAll(".point")
        .attr("cx", function(d){
            var date = dateParser(d.date);
            return x(date);
        })
        .attr("cy", function(d){
            return y(d.value);
        });
    this.selectAll(".area")
        .data([params.data])
        .exit()
        .remove();
    this.selectAll(".trendline")
        .data([params.data])
        .exit()
        .remove();
    this.selectAll(".point")
        .data(params.data)
        .exit()
        .remove();
}

plot.call(chart, {
    data: data,
    axis: {
        x: xAxis,
        y: yAxis
    }
});
</script>
</body>
</html>

```

Criando um Scatter Plot Interativo

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Visualização de Dados com D3.js</title>
  <link rel="stylesheet" href="main.css">
  <script type="text/javascript" src="d3.min.js"></script>
</head>
<body>
<script type="text/javascript" src="pesquisa.js"></script>
<script>

var w = 800;
var h = 450;

var margin = {
  top: 60,
  bottom: 80,
  left: 100,
  right: 80
};

var width = w - margin.left - margin.right;
var height = h - margin.top - margin.bottom;

var svg = d3.select("body").append("svg")
  .attr("id", "chart")
  .attr("width", w)
  .attr("height", h);

var chart = svg.append("g")
  .classed("display", true)
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

var colorScale = d3.scale.category10();

var x = d3.scale.linear()
  .domain(d3.extent(data, function(d){
    return d.idade;
  }))
  .range([0,width]);

var y = d3.scale.linear()
  .domain([1,5])
  .range([height,0]);

var tickValues = [18,25,32,39,46,53,60,67,74];

var xAxis = d3.svg.axis()
  .scale(x)
  .tickValues(tickValues)
  .orient("bottom");

var xGridlines = d3.svg.axis()
  .scale(x)
  .tickValues(tickValues)
  .tickSize(-height,-height)
  .tickFormat("")
  .orient("bottom");

var yAxis = d3.svg.axis()
  .scale(y)
  .ticks(5)
  .tickSize(20)
  .tickFormat(function(d){
    return d.toFixed(1);
  })
  .orient("left");

var yGridlines = d3.svg.axis()
  .scale(y)
  .tickSize(-width,0,0)
```

```

                .tickFormat("")
                .orient("left");

var respostascale = d3.scale.linear()
                .domain(d3.extent(data, function(d){
                    return d.respostas;
                }))
                .range([2,15]);

function drawAxis(params){
    if(params.initialize){
        this.append("g")
            .classed("gridline x", true)
            .attr("transform", "translate(0," + height + ")")
            .call(params.axis.gridlines.x);
        this.append("g")
            .classed("gridline y", true)
            .attr("transform", "translate(0,0)")
            .call(params.axis.gridlines.y);
        this.append("g")
            .classed("axis x", true)
            .attr("transform", "translate(0," + height + ")")
            .call(params.axis.x);
        this.append("g")
            .classed("axis y", true)
            .attr("transform", "translate(0,0)")
            .call(params.axis.y);
        this.select(".y.axis")
            .append("text")
            .classed("y axis-label", true)
            .attr("transform", "translate(" + -56 + "," + height/2 + ") rotate(-90)")
            .text("Avaliação (1 = Baixa, 5 = Alta)");
        this.select(".x.axis")
            .append("text")
            .classed("x axis-label", true)
            .attr("transform", "translate(" + width/2 + "," + 48 + ")")
            .text("Idade do Entrevistado");
        this.append("g")
            .append("text")
            .classed("chart-header", true)
            .attr("transform", "translate(0,-24)")
            .text("");
    }
}

function plot(params){
    drawAxis.call(this, params);
    var self = this;
    var donuts = d3.keys(params.data[0]).filter(function(d){
        return d !== "idade" && d !== "respostas";
    });

    this.selectAll(".donut")
        .data(donuts)
        .enter()
            .append("g")
            .attr("class", function(d){
                return d;
            })
            .classed("donut", true);

    this.selectAll(".donut")
        .style("fill", function(d,i){
            return colorScale(i);
        })
        .on("mouseover", function(d,i){
            d3.select(this)
                .transition()
                .style("opacity",1)
        })
        .on("mouseout", function(d,i){
            d3.select(this)
                .transition()
                .style("opacity",0.1)
        });

    donuts.forEach(function(donut){
        var g = self.selectAll("g."+donut);
    });
}

```

```

        var arr = params.data.map(function(d){
            return {
                key: donut,
                value: d[donut],
                idade: d.idade,
                respostas: d.respostas
            };
        });

        g.selectAll(".response")
            .data(arr)
            .enter()
                .append("circle")
                .classed("response", true);

        g.selectAll(".response")
            .attr("r", function(d){
                return respostascale(d.respostas);
            })
            .attr("cx", function(d){
                return x(d.idade);
            })
            .attr("cy", function(d){
                return y(d.value);
            })
            .on("mouseover", function(d,i){
                var str = " Fabricante: " + d.key + " --> ";
                str += " Idade: " + d.idade + " ";
                str += "- Respostas: " + d.respostas + " ";
                str += "- Avaliação Média: " + d.value;
                d3.select(".chart-header").text(str);
            })
            .on("mouseout", function(d,i){
                d3.select(".chart-header").text("");
            })

        g.selectAll(".response")
            .data(arr)
            .exit()
            .remove();
    });
}

plot.call(chart, {
    data: data,
    axis: {
        x: xAxis,
        y: yAxis,
        gridlines:{
            x: xGridlines,
            y: yGridlines
        }
    },
    initialize: true
})
</script>
</body>
</html>

```

Quizz

O D3.js é uma biblioteca que permite manipular dados gerando gráficos através de HTML, SVG e CSS.

O D3 não é um pacote de gráficos, mas uma ferramenta para vincular itens de dados com elementos DOM (Document Object Model) e associar atributos de dados com propriedades visuais dos elementos DOM (Document Object Model). Isso pode soar abstrato, mas isso é tudo que precisamos para criar quase qualquer gráfico.

Vincular dados a elementos DOM nos permite criar desde gráficos de barras a mapas interativos seguindo padrões semelhantes.

Criado pelo W3C o SVG (Scalable Vectorial Graphics), que em português significa Vetor Gráfico Redimensionável, é nada mais que um arquivo XML que contém tags específicas para gerar uma imagem vetorizada na sua aplicação. Com tags bastante simples você consegue gerar imagens de alta qualidade vetorizadas que por mais que você altere as proporções na tela essa não perderá qualidade, por ser uma imagem vetorizada.

Existem dois tipos principais de arquivos de imagens. Vetores e Bitmaps. Os arquivos do tipo vetor (como AI, EPS, CDR e SVG) são linhas, curvas e formas geométricas descritas matematicamente. Já os arquivos bitmaps (como JPG, PNG, GIF etc) são compostos por um grid de pixels.

9.8. Visualização de Dados com Tableau

Tableau Desktop e Tableau Public

Tableau é uma plataforma de visualização de dados. Com ele, é possível alcançar um incrível nível de descoberta de dados, análise de dados e, o mais importante, storytelling.

O tableau utiliza o VizQL – Visual Query Language, uma linguagem de consulta, inteiramente visual.

Com o tableau é possível conectar a diferentes fontes de dados, mais de uma simultaneamente. É possível conectar a dados armazenados localmente ou na nuvem, como Big Data, banco de dados SQL, uma planilha, ou aplicativos da nuvem, Google Analytics, Salesforce e Apache Hadoop.

Dimensões ou Dimensions são as variáveis categóricas. Measures ou Medidas são as variáveis numéricas.

Mini-Projeto 4 – Construindo uma Visualização de Dados para Apresentação Executiva

Especificação

Projeto – Construindo uma Visualização de Dados para Apresentação Executiva

Uma boa apresentação é aquela que conecta cada espectador através da clareza, organização de pensamento e fácil compreensão. Mas o que fazer para causar impacto? Como fazer com que o espectador sintam-se conectado ao assunto? Quais os itens que devemos prestar atenção ao preparar uma apresentação?

Há pessoas que têm o dom para falar em público, elas conseguem envolver a plateia de uma forma surreal. Um exemplo foi Steve Jobs, onde cada apresentação era um show. Mas diferente de Steve Jobs, você não precisa que sua apresentação seja um show. Precisa apenas que sua mensagem seja transmitida com sucesso. O melhor conselho para fazer uma boa apresentação é mantê-la curta, simples, clara, interessante e real.

Redução

Geralmente as apresentações longas possuem muitas repetições e informações desnecessárias, porém as pessoas querem ouvir o que é importante e relevante. E é isso que devemos fazer em nossas apresentações. Tente isso: depois de preparar sua apresentação, reduza em 30% concentrando apenas os tópicos centrais e no máximo utilize 1 slide por minuto.

Simplicidade

Significa organizar e simplificar. Comece pelo tema da sua apresentação, onde você deve resumir em uma frase. A partir disso, organize suas ideias em 3 ou 4 pontos. Se preferir faça desses pontos perguntas, como por exemplo:

Que despesas podemos reduzir?

Que despesas podemos eliminar?

Junto com as questões utilize imagens sobre o assunto, faça uso do visual. É aconselhável uma divisão 50% – 50% de textos e imagens. As imagens complementam o que está escrito.

Clareza

Não tente impressionar a plateia com palavras técnicas, isso não funciona. Use termos que possam ser compreendidos facilmente. Não insira muitos textos ou gráficos, mantenha apenas o essencial, afinal você contará a história dos dados. Qualquer material utilizado (PowerPoint, flipchart, quadro etc) é de apoio e não deve ser mais importante que você.

Envolvimento

Você tem que envolver a plateia durante sua apresentação. E para isso há diversas técnicas que podem ser utilizadas. Nelas estão inclusas repetição de impacto, perguntas retóricas, metáforas e visualização de fatos. Não apresente itens na apresentação um após o outro. Entre eles faça uma breve explicação sobre tal item ou use uma figura para ilustrar o que foi dito. Apresentações não são como listas de compras no supermercado. Não ser que esteja apresentando um check list. Faça perguntas, olhe para as pessoas, peça opiniões, dê a oportunidade para que participem da apresentação.

Realidade

Mantenha a apresentação o mais próximo da realidade. Se começar a falar sobre coisas que as pessoas não entendem e nem fazem ideia, elas não prestarão atenção em você. Utilize um estudo de caso, para que a plateia fique envolvida com a situação e a vivencie.

Enfim, trabalhe com suas habilidades. Se você não se sente confortável em contar alguma piada, não conte uma piada. Se você gosta de palavras simples, mantenha-as simples. Não tente se afastar do seu jeito natural de se comunicar.

Quizz

No Tableau, é possível importar dados do R, SAS e SPSS.

A versão free do Tableau não permite importar dados de bancos de dados.

Show me é o recurso que permite mudar de um gráfico para outro em apenas 1 clique.

O Tableau permite criar um perfil público que pode ser usado para montar um portfólio.

O Tableau Desktop pode se conectar com o Tableau Server, criando uma estrutura de servidor de visualizações.

9.9. Visualização de Dados com Qlik Sense

Conhecendo o Qlik Sense

Mais do que uma ferramenta de visualização, o Qlik Sense é uma plataforma para análise de dados.

Com o Qlik Sense é possível tomar decisões de forma colaborativa, em qualquer dispositivo, sendo ideal para grupos pequenos e departamentos.

Parte da família de produtos Qlik Sense, o Qlik Sense Desktop é um software de análise, que oferece ao usuário a possibilidade de criar visualizações de dados, de forma interativa e personalizada, e de gerar relatórios e dashboards a partir de múltiplas fontes de dados e com muita facilidade.

O Qlik Sense Desktop é gratuito, para uso pessoal e comercial, se integra com várias fontes de dados, é fácil de usar e é um aplicativo para plataforma Windows.

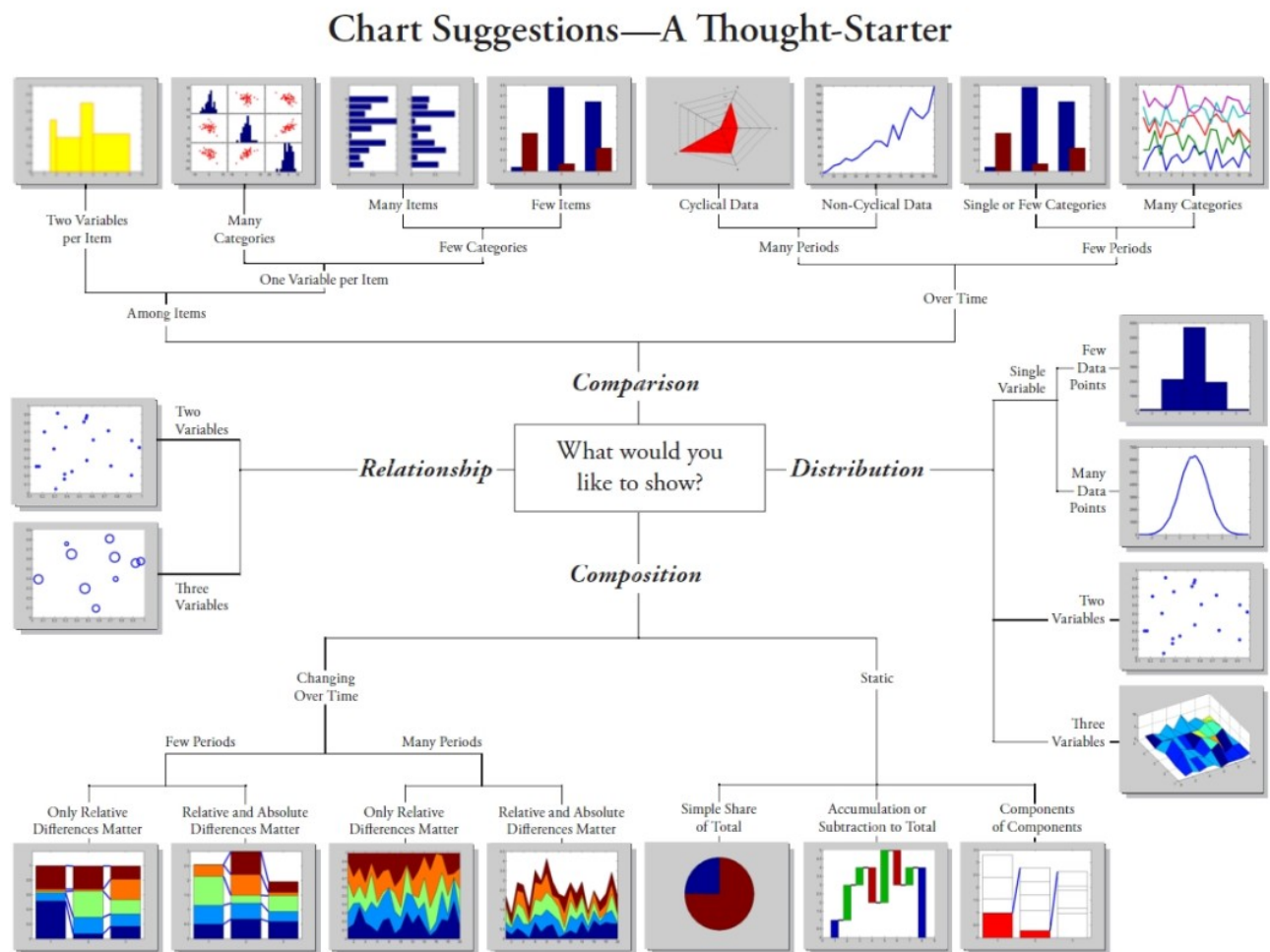
Caso seu sistema operacional não seja Windows, é possível utilizar o Qlik Sense Cloud, trabalhando apenas com o seu browser, a partir de qualquer sistema operacional.

O Qlik Sense gera visualização das informações em tempo real e não exige relatórios pré-definidos e estáticos, ou que você dependa de outros usuários. A cada clique, o Qlik Sense responde instantaneamente, atualizando todas as visualizações e exibições no aplicativo, com um conjunto recém-calculado de dados e visualizações específicas, para suas seleções.

Seu uso pode ser interessante durante o processo de análise dos dados ou na apresentação dos seus resultados.

9.10. Outras Ferramentas de Visualização

Sugestões de Gráficos



Outras Ferramentas de Visualização

- Chart.js
 - Biblioteca javascript, simples e flexível, totalmente gratuito, possui 8 tipos de gráficos, compatível com HTML5 e responsivo (gráfico se ajusta ao tamanho da tela).
- Leaflet
 - Biblioteca javascript, opensource, que permite construir mapas.
- Datawrapper
 - Produto que permite construção de gráficos, via web.
- Dygraphs
 - Baseada em javascript, opensource, é uma excelente ferramenta para construção de gráficos de séries temporais, interativos.

- Highcharts
 - Permite construção de gráficos on-line, baseado em javascript.
- Google Charts
- Polymaps
 - Baseado em javascript e OpenStreetMap, opensource, permite construção de mapas no formato SVG.
- Weka
 - Baseado em Java, permite realizar mineração de dados, construir alguns modelos de Machine Learning e alguns gráficos. Gratuita e permite trabalhar com Big Data.