## Casos de Análise Exploratória com R e Python

Nesta aula você entenderá:

* Como criar Análises Exploratórias de Dados com R e Python
* Como empregar Seleções e Visualizações dos dados para responder perguntas e verificar hipóteses sobre os dados
* O que é e como verificar correlações entre variáveis numéricas e explorar relações entre variáveis categóricas com tabelas de contingência

# Introdução

Nesta aula você vai empregar alguns conjuntos de dados para fazer Análises Exploratórias sobre esses dados empregando o ferramental que você já aprendeu até aqui. Como você verá a Análise Exploratória é um caminho bem aberto de exploração dos dados em que você busca responder e verificar hipóteses sobre os dados que você analisa. Cada conjunto de dados apresenta perguntas e hipóteses próprias. Desse modo, cada Análise é um caso bastante específico e podem diferir bastante um do outro. Vamos dar alguns exemplos aqui e você poderá buscar outros nas referências.

# Análise Exploratória dos Dados

Vamos recordar onde se localiza a Análise Exploratória de Dados (**EDA, Exploratory Data Analysis**) no contexto geral de um processo de Ciência ou Análise de Dados mais completo.

Dentro do modelo CRISP-DM a EDA compreende as fases de Entendimento e Preparação dos Dados, incluindo também a fase de Modelagem que representamos abaixo de modo parcial para indicar a existência de outros modelos mais elaborados.

Figura 5 – CRISP DM e onde se localiza a EDA

Diagrama

Descrição gerada automaticamente

**Fonte: Adaptado de Chapman (2000) pelo autor**

A EDA é usada, então, para analisar e investigar conjuntos de dados e resumir suas características principais, empregando métodos quantitativos e de visualização dos dados.

Ela irá permitir responder a perguntas do tipo *O que acontece?* e *Por que Acontece*? trazendo um conhecimento sobre os dados e você poderá fazer boa parte disso a partir de seleções e visualizações dos dados empregando o ferramental que você já aprendeu aqui. Esses resultados por si só já serão um grande resultado e já permitem tomarmos uma série de decisões baseadas em dados mesmo sem modelos mais elaborados como modelos de predição (se necessário volte na Aula 1 onde falamos um pouco mais sobre isso antes).

Para essas análises são empregados vários conceitos que você já aprendeu aqui, como conceitos de Probabilidade e Estatística (estatísticas descritivas dos dados, distribuições de probabilidade, aplicação de probabilidades condicionais etc.) e técnicas de seleção e transformação dos dados (seleção de instâncias e atributos, agregações, transformações de formato, codificações de valores, criação de atributos derivados, merge de dados etc.).

# Método da Análise Exploratória

A **Análise Exploratória de Dados (EDA)** é uma abordagem para análise de dados que emprega uma variedade de técnicas, incluindo seleções e transformações dos dados e técnicas de visualização para, dentre outras:

1. maximizar a percepção dos dados;
2. descobrir a estrutura dos dados;
3. identificar variáveis ​​importantes;
4. identificar valores mais frequentes;
5. detectar outliers e anomalias;
6. verificar a qualidade dos dados;
7. identificar padrões; etc.

EDA é um **ciclo iterativo**, de refinamento sucessivo de análise:

1. Crie perguntas ou hipóteses sobre os dados

* *ex. Será que a maior parte dos brasileiros têm renda inferior à renda mínima para declaração de IR?*

1. Busque as respostas visualizando, transformando e modelando os dados

* *ex. Você pode obter os quartis de renda ou ainda fazer um gráfico de distribuição dos valores de renda e confrontar com valor mínimo do IR*

1. Use o que você aprendeu para refinar as perguntas e as respostas

* *ex. Suponha que você chegue a resposta de que 80% dos brasileiros não pagam IR (em 2014 eram apenas 15% dos brasileiros que declaravam IR!). Você pode, então, querer refinar a sua busca para buscar esse percentual por região ou estado.*

O EDA **não é um processo formal** com um conjunto regras pré-definido. O EDA é uma abordagem que, antes de tudo, pressupõe um espírito ou postura crítica sobre os dados. Um espírito quase científico ou investigativo. Como todo processo investigativo **alguns caminhos de análise serão produtivos, outros não**. Os caminhos produtivos podem levar a novas análises, comunicação de resultados ou mesmo a tomada de decisões. Os caminhos infrutíferos são lições aprendidas sobre os dados e também registrarão hipóteses rejeitadas sobre os dados.

Esse *espírito investigativo sobre os dados* é mais modernamente representado pela expressão **data thinking**. Nessa abordagem a EDA se diferencia da análise propriamente estatística que busca caracterizar os dados por gráficos e dados estatísticos. Na EDA a ideia predominante é deixar que os dados revelem sua estrutura e seus padrões, produzindo poderosos **insights** sobre os dados.

**Exploratory Data Analysis, Tukey, (1977)** é em geral considerado o texto seminal da EDA, e Tukey faz uma série de contribuições à análise de dados que vigoram até hoje (como a rejeição dos gráficos de pizza ou sua criação dos gráficos de caixa).

“Muito melhor uma resposta aproximada para a pergunta certa, que muitas vezes é vaga, do que uma resposta exata para a pergunta errada, que sempre pode ser precisa”. - John Tukey

# Quando aplicar a EDA

Embora possa diferir em termos de profundidade a **EDA é uma fase importante de qualquer análise de dados, das mais simples em que você busca respostas diretas sobre os dados (qual a média de idade dos desempregados?), até nas análises mais sofisticadas em que são empregados modelos de Aprendizado de Máquina**. Mesmo nos modelos mais simples você sempre precisa investigar a qualidade de seus dados, ou ainda fazer operações de seleção ou limpeza de dados para empregar somente dados que atendem à sua pesquisa.

O objetivo da EDA é criar uma compreensão de seus dados e a maneira mais direta de fazer isso é empregar perguntas sobre os dados. Elas irão, então, orientar a investigação definindo a exploração de um certo aspecto dos dados e ajudando a decidir que parte dados será empregada, quais gráficos, e os modelos ou transformações necessárias. Não há regras de que perguntas devem ser feitas e elas podem variar muito em razão do contexto dos dados. Mas os itens no início dessa seção dão uma dica de que tipo de perguntas devemos fazer e ainda podemos perguntar como nossos dados variam, como duas variáveis se relacionam, comparar períodos diferentes dos dados etc.

# Técnicas de Análise Exploratória de Dados

A EDA emprega métodos quantitativos, como estatísticas e sumarizações dos dados, e técnicas de visualização dos dados para obter uma série de análise. Em geral elas envolvem:

* **Análise univariada dos dados** que incluem a análise de cada atributo dos dados individualmente a partir de estatísticas dos dados, análise da distribuição dos valores, identificação de anomalias e padrões etc.
* **Análise bivariadas e multivariadas** que avaliam a relação entre cada variável no conjunto de dados com uma variável objetivo, ou entre várias variáveis.
* **Análise de Agrupamentos**, que incluem a redução de dimensão dos dados em grupos de dados que se assemelham e se diferenciam dos demais

# Um Roteiro para a EDA

Empregando técnicas de natureza gráfica e algumas técnicas quantitativas você pode fazer o seguinte roteiro fazendo perguntas a associadas a cada item sobre os dados:

1. Explore a estrutura dos dados
2. Examine então dados brutos e atributos relevantes
3. Gere estatísticas simples sobre os dados
4. Verifique os valores possíves dos atributos, em especial atributos categóricos
5. Responda a perguntas relevantes ou de interesse sobre os dados
6. Dependendo do problema você pode então analisar:
7. Evolução ou Tendência dos Dados
8. Distribuições dos Dados
9. Comparativos de Quantidades e Proporções
10. Correlações e Relações entre os Dados

Este não é, claro, a única forma de fazer um EDA. Mas na ausência de um roteiro melhor esse pode ser um bom começo.

# Estudos de Caso

A seguir faremos algumas análises sobre diferentes conjuntos de dados seguindo alguns princípios da EDA (processo informal, interativo, de livre exploração dos dados etc.) aplicando técnicas quantitativas e de visualização já estudadas, e outras novas, para que você entenda melhor como criar suas próprias análises.

*Recursos novos empregados aqui podem ser consultados na documentação dos pacotes. Eles permitem introduzir novos exemplos de recursos que não empregamos antes mas, principalmente, mostram que existem muitos mais recursos que você pode buscar para construir suas próprias análises. Assim tenha em mente nesta aula muito mais o tipo de análise e como técnicas de forma genérica são empregadas (como normalização, aplicação de linhas de tendência ou ainda correlogramas) do que propriamente a codificação, que poderá ser buscada na referência dos pacotes havendo muitas diferentes alternativas de implementação.*

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

# Caso: Índices da Bolsa de Valores, *Séries Temporais*

Quando falamos de Evolução dos dados estamos interessados em observar a evolução, o comportamento, de uma ou mais variáveis dependentes, com relação a uma outra. Você pode estar, por exemplo, interessado em saber como evolui a quantidade de energia produzida por uma turbina à medida que aumenta o fluxo de água, ou como aumenta a acidez de um vinho a medida que a temperatura aumenta. Mas um caso muito frequente é querermos observar a **evolução de variáveis com relação ao tempo**. Chamamos esse conjunto de dados de **série temporal**. Existem muitos casos de interesse de séries temporais e, por isso, vale a pena explorarmos um desses casos aqui. As séries temporais requerem todo um ferramental estatístico próprio e há uma série de aspectos que não trataremos (identificação de sazonalidades, periodicidade etc.). Mas poderemos verificar o que nosso ferramental até aqui já pode nos proporcionar para a análise desses dados e entendermos também algumas das dificuldades de lidarmos com esse tipo de dado.

## Exploração Inicial dos Dados

Devemos começar entendendo a estrutura dos dados que iremos trabalhar. Quantidade de dados, significado dos atributos, formatos, são perguntas essenciais para extrair informações dos dados.

bovespa = pd.read\_csv('http://meusite.mackenzie.br/rogerio/TIC/bovespa.csv')  
  
display(bovespa.shape)  
display(bovespa.head())  
display(bovespa.tail())  
display(bovespa.Stock.value\_counts())  
display(bovespa.data.duplicated().sum())  
display(bovespa.data.min(),bovespa.data.max())

(129, 3)

data Stock value  
0 2020-01-02 VALE3 13.45  
1 2020-01-03 VALE3 13.29  
2 2020-01-06 VALE3 13.14  
3 2020-01-07 VALE3 13.23  
4 2020-01-08 VALE3 13.22

data Stock value  
124 2020-03-03 DOLAR 4.4724  
125 2020-03-04 DOLAR 4.5132  
126 2020-03-05 DOLAR 4.5834  
127 2020-03-06 DOLAR 4.6062  
128 2020-03-09 DOLAR 4.5898

PETR4 43  
DOLAR 43  
VALE3 43  
Name: Stock, dtype: int64

86

{"type":"string"}

{"type":"string"}

O significado dos dados aqui um tanto evidente, mas em muitos casos reais é necessário que sejam fornecidas informações sobre o conteúdo dos dados, *metadados*, como seu significado, unidade empregada etc. Essas informações são dependentes de cada caso e vamos assumir que essas informações, como no nosso exemplo, já são conhecidas.

A rápida inspeção dos dados acima mostra que temos valores de três índices da bolsa para o período de 2020-01-02 a 2020-03-09.

## Reshape dos Dados

Mas a inspeção acima também mostra que os dados não estão exatamente no formato que podíamos esperar: as datas se repetem e os códigos dos índices também. O ideal é que os dados estivessem organizados de forma que cada variável fosse uma coluna e cada observação (ou caso) uma linha. É o que chamamos de **Tidy** (algo equivalente a 3a forma normal de bancos de dados relacionais) e que você poderá consultar mais sobre isso nas referências ao final.

Essa é uma situação muito comum e comumente, em casos reais, você precisará fazer o *reshape* dos dados para um formato mais adequado.

O Pandas fornece várias instruções para reshape de dados como merge, pivot, melt etc. Como falamos antes não demos muito atenção para essas funções, mas elas podem ser facilmente consultadas na documentação do Pandas ou no guia rápido. Aqui podemos empregar a instrução pivot para reorganizar os dados no formato *Tidy* data.

bovespa = pd.pivot(bovespa,columns=['Stock'],values='value',index='data').reset\_index()   
bovespa = bovespa.rename\_axis('',axis='columns') # este rename é opcional   
bovespa.head()

data DOLAR PETR4 VALE3  
0 2020-01-02 4.0163 16.270000 13.45  
1 2020-01-03 4.0234 15.990000 13.29  
2 2020-01-06 4.0570 16.219999 13.14  
3 2020-01-07 4.0604 16.059999 13.23  
4 2020-01-08 4.0662 15.700000 13.22

## Diferentes Fontes de Dados

Mais comum ainda é a necessidade de juntarmos dados de diferentes fontes. Como você pode entender isso irá variar bastante de caso a caso, mas aqui podemos fazer uma junção simples de dados e adicionar aos dados um novo índice.

IBOV = pd.read\_csv('http://meusite.mackenzie.br/rogerio/TIC/IBOV.csv')  
  
display(IBOV.shape)  
display(IBOV.head())  
display(IBOV.data.min(),bovespa.data.max())

(43, 3)

data Stock value  
0 2020-01-02 IBOV 118573.0  
1 2020-01-03 IBOV 117707.0  
2 2020-01-06 IBOV 116878.0  
3 2020-01-07 IBOV 116662.0  
4 2020-01-08 IBOV 116247.0

{"type":"string"}

{"type":"string"}

O Pandas também fornece várias formas para combinarmos dados (concat, merge etc.) e aqui talvez pudéssemos apenas adicionar a coluna de valores do índice ao nosso dado inicial. Mas vamos uma *junção* dos dados, algo que pode ser útil para você em muitos casos práticos. A junção funciona de forma análoga a um *join* da SQL.

IBOV = IBOV.rename(columns={'value':'IBOV'})

bovespa = pd.merge(bovespa, IBOV[['data','IBOV']], on='data')  
display(bovespa.head())

data DOLAR PETR4 VALE3 IBOV  
0 2020-01-02 4.0163 16.270000 13.45 118573.0  
1 2020-01-03 4.0234 15.990000 13.29 117707.0  
2 2020-01-06 4.0570 16.219999 13.14 116878.0  
3 2020-01-07 4.0604 16.059999 13.23 116662.0  
4 2020-01-08 4.0662 15.700000 13.22 116247.0

## Transformações dos Dados

Muitas transformações também podem ser necessárias para tornar o dado mais adequado para uma análise e são comuns transformações de mudança de tipos, *encode* de dados, formatos etc. como transformações mais elaboradas como a normalização dos dados que faremos mais adiante.

Neste ponto do nosso exemplo, basta observar que o tipo de dado de 'data', embora apresente os valores de forma correta, não é o tipo de dado *ideal* para esse tipo de informação. Você por exemplo não poderá com esse tipo de dado fazer uma operação de diferença de datas ou obter o dia da semana. Vamos então transformar esse atributo para o formato mais adequado de *datetime*.

display(bovespa.dtypes)  
bovespa.data = pd.to\_datetime(bovespa.data, format='%Y-%m-%d',errors='coerce')  
display(bovespa.dtypes)

data object  
DOLAR float64  
PETR4 float64  
VALE3 float64  
IBOV float64  
dtype: object

data datetime64[ns]  
DOLAR float64  
PETR4 float64  
VALE3 float64  
IBOV float64  
dtype: object

## Qualidade dos Dados

A análise de dados ausentes, *outliers* (dados discrepantes) dentre outras, permitem termos uma ideia da qualidade dos dados e são essenciais antes de prosseguirmos na análise.

print(bovespa.isnull().sum())  
print(bovespa.describe())

data 0  
DOLAR 0  
PETR4 0  
VALE3 0  
IBOV 0  
dtype: int64  
 DOLAR PETR4 VALE3 IBOV  
count 43.000000 43.000000 43.000000 43.000000  
mean 4.265325 14.243721 12.046279 113382.767442  
std 0.158326 1.675228 1.287173 6520.692382  
min 4.016300 7.260000 7.970000 86067.000000  
25% 4.164300 14.205000 11.790000 113765.500000  
50% 4.242200 14.630000 12.050000 115528.000000  
75% 4.354650 14.915000 13.190000 116689.000000  
max 4.606200 16.270000 13.630000 119528.000000

Neste exemplo simples nos limitamos a verificar que não existem valores nulos nos dados e algumas de suas estatísticas.

## Explorando os Dados

A partir daí estamos mais ou menos prontos para analisar questões específicas dos dados. Dependendo do problema você pode querer analisar: evoluções ou tendência dos dados, distribuições dos dados, analisar quantidades e proporções, relações etc. Neste exemplo vamos nos concentrar na análise de evolução dos índices da bolsa no período o que nos permitirá aprender e praticar algumas técnicas bastante importantes e comuns.

### Linha de Tendência Linear

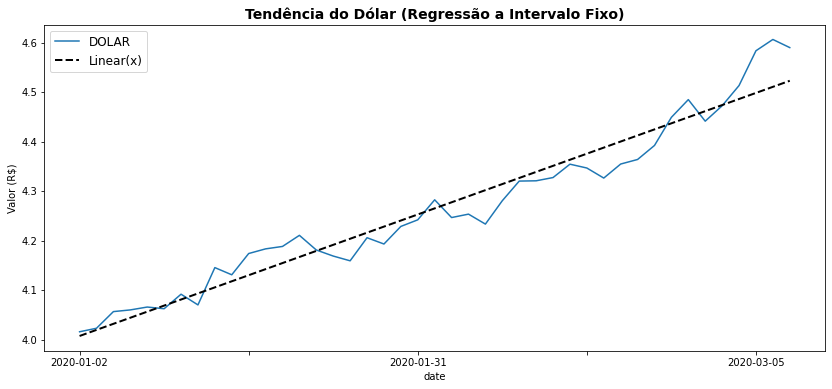
Aqui você vai encontrar um problema muito comum ao buscarmos tendências em dados temporais. Sua variável independente (a linha do tempo) não pode ser aplicada diretamente a uma regressão linear (ou outra qualquer) pois seus dados não são numéricos. Para fazer isso teremos de empregar um artifício e transformar a linha do tempo em um atributo numérico.

Isso pode ser feito atribuindo-se para cada data do conjunto de dados um valor inteiro diferente a partir de 0. Se cada linha do conjunto de dados corresponde a uma data, esse valor pode ser o próprio índice do seu conjunto de dados (para um DataFrame, df.index).

bovespa.index

Int64Index([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,  
 34, 35, 36, 37, 38, 39, 40, 41, 42],  
 dtype='int64')

fig, ax = plt.subplots(figsize=(14,6))  
  
ax.plot(bovespa.index, bovespa.DOLAR, label='DOLAR')  
  
ax.set\_title('Tendência do Dólar (Regressão a Intervalo Fixo)', fontsize=14, weight='bold')  
ax.set\_xlabel("date")  
ax.set\_ylabel("Valor (R$)")  
  
z = np.polyfit(bovespa.index,bovespa.DOLAR, 1)  
p = np.poly1d(z)  
ax.plot(bovespa.index,p(bovespa.index), linestyle='dashed', label='Linear(x)' ,linewidth=2, color='k')  
  
# OU  
# x = np.arange(0,len(bovespa),1)  
# z = np.polyfit(x,bovespa.DOLAR, 1)  
# p = np.poly1d(z)  
# ax.plot(bovespa.data,p(x), linestyle='dashed', label='Linear(x)' ,linewidth=2, color='k')  
  
ax.legend(fontsize=12)  
  
import matplotlib.ticker as ticker  
ax.xaxis.set\_major\_locator(ticker.MaxNLocator(5))  
ax.set\_xticklabels(['',bovespa.iloc[0].data.date(),'',bovespa.iloc[20].data.date(),'',bovespa.iloc[40].data.date()])  
  
plt.show()



Não se prenda muito aos detalhes do gráfico, o que queremos que você entenda é a forma geral e alguns problemas que você deverá lidar quando trabalha com séries temporais. Os detalhes você poderá deixar para quando estiver trabalhando com o caso específico seu.

O procedimento acima é útil para em muitos casos que envolvem séries temporais. Acima empregamos o índice dos dados (43 valores de para cada data) para linha de tempo da regressão mas, na ausência de um índice, podemos criar um valor diferente para cada data empregando, por exemplo:

x = np.arange(0,len(bovespa),1)

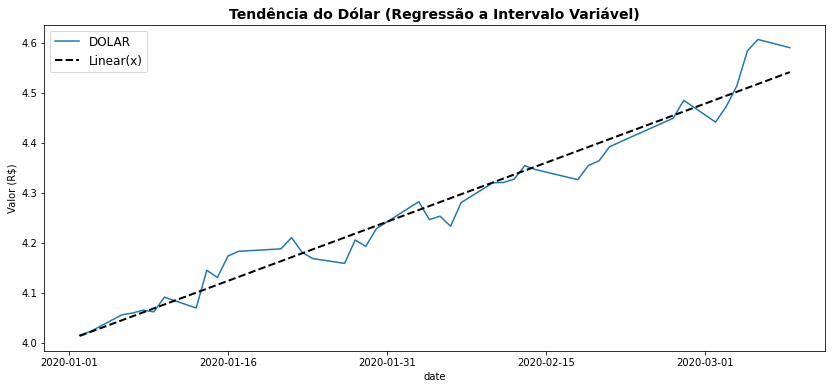
isso encontra-se comentado no código acima como uma forma opcional.

Há, entretanto, uma série de detalhes nisso. Por exemplo, você pode ter intervalos de datas diferentes e, neste caso, pode ser necessário associar a cada data a distância com relação a menor data no seu conjunto de dados (não empregando, portanto, valores com um incremento fixo como fizemos).

No nosso exemplo, como os dados da bolsa referem-se a dias úteis há intervalos de datas maiores nos finais de semana e feriados e, embora os resultados sejam praticamente os mesmos aqui, a rigor precisamos empregar intervalos de datas diferentes e para muitos casos isso pode ser realmente necessário. Fazemos esse procedimento abaixo adicionando um novo atributo com quantidade de dias a partir da data inicial. A função dt.days permite transformar esses valores para o cálculo da regressão.

bovespa['dias'] = bovespa['data'] - bovespa['data'].min()

fig, ax = plt.subplots(figsize=(14,6))  
  
ax.plot(bovespa.data, bovespa.DOLAR, label='DOLAR')  
  
ax.set\_title('Tendência do Dólar (Regressão a Intervalo Variável)', fontsize=14, weight='bold')  
ax.set\_xlabel("date")  
ax.set\_ylabel("Valor (R$)")  
  
z = np.polyfit(bovespa.dias.dt.days,bovespa.DOLAR, 1)  
p = np.poly1d(z)  
ax.plot(bovespa.data,p(bovespa.dias.dt.days), linestyle='dashed', label='Linear(x)' ,linewidth=2, color='k')  
  
ax.legend(fontsize=12)  
  
import matplotlib.ticker as ticker  
ax.xaxis.set\_major\_locator(ticker.MaxNLocator(5))  
  
plt.show()



Aqui o resultado é praticamente o mesmo, mas para os casos onde os intervalos de data tenham diferenças maiores isso poderá ser relevante.

### Linha de Tendência Média

Uma outra forma de entendermos as linhas de tendência dos dados é empregarmos a média dos vários valores e ligar esses pontos a uma linha de tendência. Aqui não buscamos aproximar nenhum modelo prévio, como um modelo linear ou polinomial, mas exibimos simplesmente uma média de tendência dos dados.

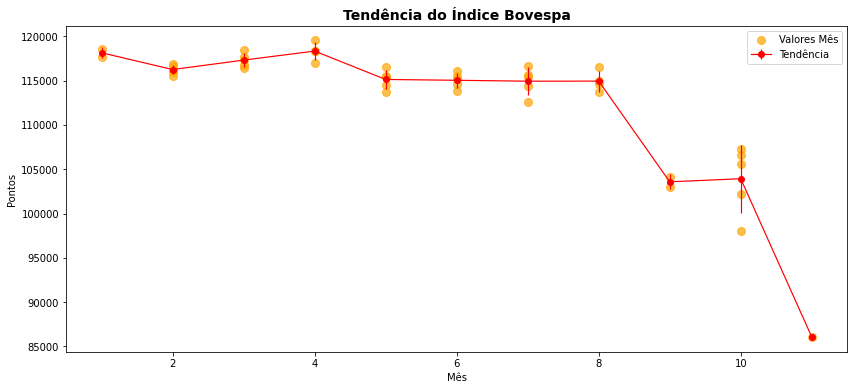
Essa linha média ainda pode ser incrementada com uma margem de erro, por exemplo o desvio padrão, permitindo acompanhar a dispersão dos dados ao longo da linha média. Para isso, no lugar do plot você poderá empregar o comando plt.errorbar. Como nossos dados apresentam apenas uma única cotação diária, iremos agregar a cotações em semanas e exibir as cotações semanais.

bovespa['week'] = bovespa['data'].dt.isocalendar().week

bovespa.head()

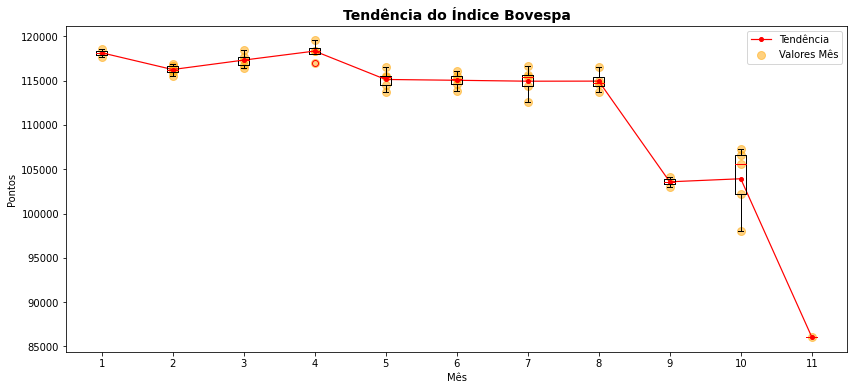
data DOLAR PETR4 VALE3 IBOV dias week  
0 2020-01-02 4.0163 16.270000 13.45 118573.0 0 days 1  
1 2020-01-03 4.0234 15.990000 13.29 117707.0 1 days 1  
2 2020-01-06 4.0570 16.219999 13.14 116878.0 4 days 2  
3 2020-01-07 4.0604 16.059999 13.23 116662.0 5 days 2  
4 2020-01-08 4.0662 15.700000 13.22 116247.0 6 days 2

fig, ax = plt.subplots(figsize=(14,6))  
  
yerr = []  
ymean = []  
for w in bovespa.week.unique():  
 yerr.append(bovespa[bovespa.week == w]['IBOV'].std())  
 ymean.append(bovespa[bovespa.week == w]['IBOV'].mean())  
yerr = np.array(yerr)   
ymean = np.array(ymean)   
  
plt.errorbar(bovespa.week.unique(), ymean, yerr=yerr, label='Tendência', marker='o', color='r', linewidth=1.2)  
plt.scatter(bovespa.week, bovespa.IBOV, label='Valores Mês', marker='o', color='orange', alpha=0.7, s=64)  
  
plt.title("Tendência do Índice Bovespa", fontsize=14, weight='bold')  
plt.xlabel("Mês")  
plt.ylabel("Pontos")  
plt.legend()  
  
plt.show()



O mesmo procedimento poderia ser feito se tivéssemos cotações diárias do índice, e se já você acompanhou gráficos empregados na análise técnica de ações da bolsa na mídia ou em aplicativos de investimento, deve notar a semelhança dos gráficos. No caso de ações da bolsa, entretanto, é mais comum essa tendência ser acompanhada por gráficos de caixa, ou boxplot como abaixo.

fig, ax = plt.subplots(figsize=(14,6))  
  
plt.plot(np.array(bovespa.week.unique()), ymean, 'r-', label='Tendência', marker='o', linewidth=1.2, markersize=4)  
plt.scatter(bovespa.week, bovespa.IBOV, label='Valores Mês', marker='o', alpha=0.5, s=64, color='orange')  
for week in bovespa.week.unique():  
 plt.boxplot(bovespa[bovespa.week == week].IBOV, positions=[week], widths=0.15, flierprops=dict(color='r',markeredgecolor='r'),medianprops=dict(color='r'))  
  
plt.title("Tendência do Índice Bovespa", fontsize=14, weight='bold')  
plt.xlabel("Mês")  
plt.ylabel("Pontos")  
plt.legend()  
  
plt.show()

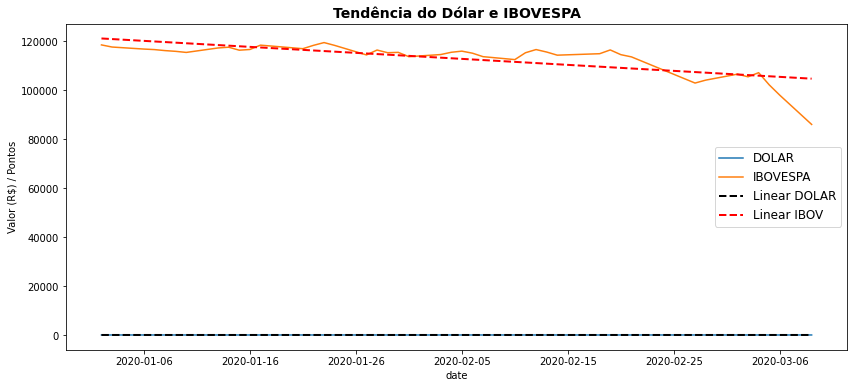


Os gráficos de caixa funcionam como uma forma sintética de exibir distribuições e você pode, por exemplo, observar a maior dispersão dos dados na semana 10.

### Normalizando os Dados: Tratando Dado em Escalas Diferentes

Se você empregar agora diretamente o gráfico anterior para analisar a tendência não só do índice Bovespa mas também um dos demais índices o resultado não irá ser dos melhores como você pode observar:

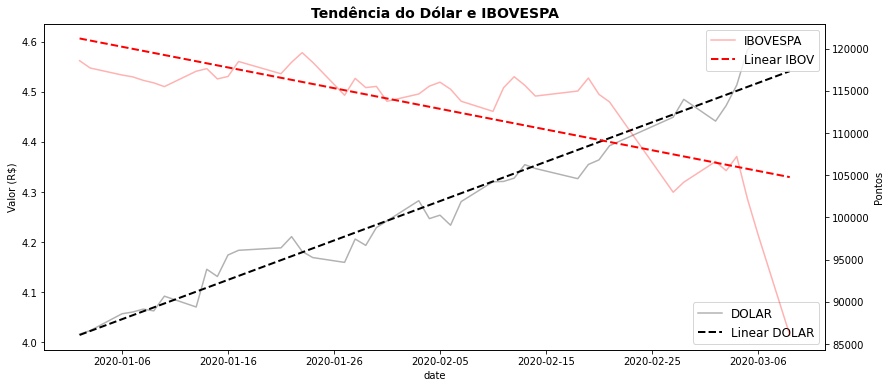
fig, ax = plt.subplots(figsize=(14,6))  
  
ax.plot(bovespa.data, bovespa.DOLAR, label='DOLAR')  
ax.plot(bovespa.data, bovespa.IBOV, label='IBOVESPA')  
  
ax.set\_title('Tendência do Dólar e IBOVESPA', fontsize=14, weight='bold')  
ax.set\_xlabel("date")  
ax.set\_ylabel("Valor (R$) / Pontos")  
  
z = np.polyfit(bovespa.dias.dt.days,bovespa.DOLAR, 1)  
p = np.poly1d(z)  
ax.plot(bovespa.data,p(bovespa.dias.dt.days), linestyle='dashed', label='Linear DOLAR' ,linewidth=2, color='k')  
  
z = np.polyfit(bovespa.dias.dt.days,bovespa.IBOV, 1)  
p = np.poly1d(z)  
ax.plot(bovespa.data,p(bovespa.dias.dt.days), linestyle='dashed', label='Linear IBOV' ,linewidth=2, color='r')  
  
ax.legend(fontsize=12)  
  
import matplotlib.ticker as ticker  
ax.xaxis.set\_major\_locator(ticker.MaxNLocator(8))  
  
plt.show()



Se você observar com atenção o mal resultado dessa visualização é por quê os dados estão em escalas de valores muito diferentes e você não consegue, no mesmo gráfico, observar claramente as variáveis do dólar que se encontram em um intervalo de valores e os valores de índice Bovespa, no intervalo .

O comando ax.twinx() no modo Artist Layer pode ser uma alternativa. Ele permite você apresentar duas escalas diferentes no mesmo gráfico e podemos assim comparar a tendência das duas séries de dados simultaneamente.

fig, ax = plt.subplots(figsize=(14,6))  
  
ax.plot(bovespa.data, bovespa.DOLAR, label='DOLAR', color='k', alpha=0.3)  
  
ax2 = ax.twinx() # instantiate a second axes that shares the same x-axis  
  
ax2.plot(bovespa.data, bovespa.IBOV, label='IBOVESPA', color='r', alpha=0.3)  
  
ax.set\_title('Tendência do Dólar e IBOVESPA', fontsize=14, weight='bold')  
ax.set\_xlabel("date")  
ax.set\_ylabel("Valor (R$)")  
ax2.set\_ylabel("Pontos")  
  
z = np.polyfit(bovespa.dias.dt.days,bovespa.DOLAR, 1)  
p = np.poly1d(z)  
ax.plot(bovespa.data,p(bovespa.dias.dt.days), linestyle='dashed', label='Linear DOLAR' ,linewidth=2, color='k')  
  
z = np.polyfit(bovespa.dias.dt.days,bovespa.IBOV, 1)  
p = np.poly1d(z)  
ax2.plot(bovespa.data,p(bovespa.dias.dt.days), linestyle='dashed', label='Linear IBOV' ,linewidth=2, color='r')  
  
ax.legend(loc='lower right',fontsize=12)  
ax2.legend(fontsize=12)  
  
import matplotlib.ticker as ticker  
ax.xaxis.set\_major\_locator(ticker.MaxNLocator(8))  
  
plt.show()



Esse procedimento, entretanto, só é útil quando temos até duas escalas diferentes de dados. Para mais de duas escalas a forma que podemos empregar é a normalização dos dados.

De forma bastante simples, a **Normalização** consiste em redimensionarmos os dados criando versões deslocadas e escalonadas das estatísticas para podermos fazer comparação dos dados. Em outras palavras a **Normalização** consiste em ajustar os valores medidos em diferentes escalas para uma escala comum e essa técnica tem um papel importante não só para a comparação de valores mas, como você verá mais adiante no curso, também em processos de aprendizado de máquina para eliminar o efeito de peso da escala entre diferentes variáveis atribuindo a cada uma o mesmo *pesos* Por exemplo, sem um ajuste uma variável de renda anual - cuja a undidade por ser de milhares ou mais - teria um peso muito maior que idade - com unidade de algumas dezenas - simplesmente por estarem em escalas diferentes.

Existem ainda diferentes formas de rescalarmos e redimensionarmos os dados, todas elas muitas vezes colocadas sobre o mesmo termo comum de *normalização*. Mas, em geral, normalização normalmente significa redimensiona os valores em um intervalo de [0,1] e outra técnicas, a estandardização, normalmente significa fazer o redimensionamento dos dados com média de 0 e um desvio padrão de 1 (variação de unidade).

*Cuidado ao empregar normalizações dos dados! Ao exibir os dados normalizados você não estará mais observando os valores reais dos dados, mas seus movimentos relativos.*

Para aplicar a normalização você pode aplicar simplesmente uma normalização *min/max*. Para dados unicamente positivos basta dividir cada valor pelo valor máximo da série de dados, o que levar cada valor para o intervalo (verifique na fórmula abaixo essa afirmação!). Na presença de valores negativos, entretanto, é necessário aplicar a fórmula:

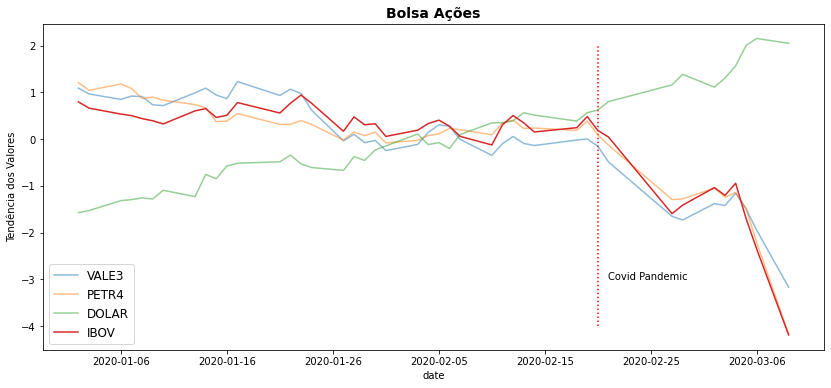
Mas aqui vamos adotar um procedimento muito usado, e igualmente simples, aplicando uma normalização conhecida como *standard scaler*, estandardização, ou que normaliza os valores levando o conjunto de dados a uma média 0 e desvio padrão 1.

Sim, essa fórmula já é familiar para você! É o que você estudou já na aula 2 e mais adiante na aula 5. O resultado de redimensionarmos o dados com essa técnica encontra-se a seguir.

bovespa\_norm = ( bovespa.drop(columns=['data','dias','week']) - bovespa.drop(columns=['data','dias','week']).mean() ) / bovespa.drop(columns=['data','dias','week']).std()  
bovespa\_norm = pd.concat([bovespa[['data','dias','week']], bovespa\_norm],axis=1)  
bovespa\_norm.head()

data dias week DOLAR PETR4 VALE3 IBOV  
0 2020-01-02 0 days 1 -1.572860 1.209554 1.090545 0.795963  
1 2020-01-03 1 days 1 -1.528018 1.042412 0.966242 0.663155  
2 2020-01-06 4 days 2 -1.315796 1.179707 0.849708 0.536022  
3 2020-01-07 5 days 2 -1.294322 1.084198 0.919628 0.502896  
4 2020-01-08 6 days 2 -1.257690 0.869302 0.911860 0.439253

fig, ax = plt.subplots(figsize=(14,6))  
  
ax.plot(bovespa\_norm.data, bovespa\_norm.VALE3, label='VALE3', alpha=0.5)  
ax.plot(bovespa\_norm.data, bovespa\_norm.PETR4, label='PETR4', alpha=0.5)  
ax.plot(bovespa\_norm.data, bovespa\_norm.DOLAR, label='DOLAR', alpha=0.5)  
ax.plot(bovespa\_norm.data, bovespa\_norm.IBOV, label='IBOV')  
  
ax.set\_title('Bolsa Ações', fontsize=14, weight='bold')  
ax.set\_xlabel("date")  
ax.set\_ylabel("Tendência dos Valores")  
  
ax.legend(fontsize=12)  
  
plt.vlines(np.datetime64('2020-02-20'),-4,2,color='r', linestyles=':')  
plt.text(np.datetime64('2020-02-21'),-3,'Covid Pandemic')  
import matplotlib.ticker as ticker  
ax.xaxis.set\_major\_locator(ticker.MaxNLocator(8))  
  
plt.show()



Não vamos tirar conclusões sobre esses dados e podemos deixar isso para os economistas e analistas de mercado. Nosso interesse aqui é que você entenda algumas técnicas análise com os recursos que já aprendemos e que poderão ser úteis a você em muitos casos. É, de qualquer modo, fácil ver que o índice da bolsa segue de forma bastante próxima os movimentos das ações da VALE3 e PETR4, e identificar a presença de vales do índice quando de fortes aumentos do dólar. Podemos identificar também a inversão dos movimentos do mercado com o início da pandemia de Covid no Brasil no começo de 2020.

# Caso: Vinhos no Mundo, *Perguntas Relevantes sobre os Dados*

Vamos agora explorar um conjunto de dados com cerca de 130.000 avaliações de Vinhos. Em geral o objetivo da análise é de responder perguntas sobre os dados, hipóteses que lançamos sobre os dados e que buscamos confirmar. Vamos, assim, tentar responder a várias perguntas relevantes e curiosas sobre esse mercado, como *Que região do Brasil produz mais vinhos dentro dos avaliados?* ou *Se os vinhos Franceses são mesmo mais caros e mais bem avaliados?*

## Exploração Inicial dos Dados

Aqui não vamos nos preocupar com diferentes fontes de dados ou transformações. O formato dos dados, como você pode verificar na rápida inspeção abaixo também parece adequada e poderemos seguir diretamente para verificar a qualidade dos dados.

wine = pd.read\_csv('http://meusite.mackenzie.br/rogerio/TIC/winemag-data-130k-v2.csv')  
  
display(wine.shape)  
display(wine.head())

(129971, 14)

Unnamed: 0 country ... variety winery  
0 0 Italy ... White Blend Nicosia  
1 1 Portugal ... Portuguese Red Quinta dos Avidagos  
2 2 US ... Pinot Gris Rainstorm  
3 3 US ... Riesling St. Julian  
4 4 US ... Pinot Noir Sweet Cheeks  
  
[5 rows x 14 columns]

## Qualidade dos Dados

Podemos observar que existem poucos vinhos nacionais avaliados e que a base possui um grande número de valores ausentes.

display( wine.country.value\_counts()[0:10] )  
print( 'Percentual de vinhos nacionais: ', wine[wine.country == 'Brazil'].country.count() / len(wine))  
display( wine.isnull().sum() / len(wine) )

US 54504  
France 22093  
Italy 19540  
Spain 6645  
Portugal 5691  
Chile 4472  
Argentina 3800  
Austria 3345  
Australia 2329  
Germany 2165  
Name: country, dtype: int64

Percentual de vinhos nacionais: 0.00040008925067899764

Unnamed: 0 0.000000  
country 0.000485  
description 0.000000  
designation 0.288257  
points 0.000000  
price 0.069215  
province 0.000485  
region\_1 0.163475  
region\_2 0.611367  
taster\_name 0.201922  
taster\_twitter\_handle 0.240154  
title 0.000000  
variety 0.000008  
winery 0.000000  
dtype: float64

Particularmente, vamos estar interessados aqui no preço dos vinhos e a presença de valores ausentes é algo que você precisará ter em mente ao produzir estatísticas, como média e desvio padrão, desse dado. Nos casos em que você emprega os dados em algoritmos de aprendizado pode ser até necessário você *tratar* os valores ausentes eliminando as linhas, os atributos ou mesmo imputando valores.

## Explorando Perguntas Relevantes sobre os Dados

Vamos agora fazer uma série de perguntas sobre os dados e buscar respondê-las a partir de seleções ou visualizações dos dados. Veja que muitas dessas perguntas funcionam como hipóteses que fazemos sobre os dados. Por exemplo, ao perguntar *de que regiões vem os vinhos Brasileiros?* supomos haver regiões específicas de produção que queremos descobrir, ou ainda quando perguntamos *Os melhores vinhos são os vinhos mais caros?* estamos buscando confirmar ou rejeitar essa hipótese sobre os dados (embora existam técnicas mais rigorosas de análise, como os *testes de hipóteses*, e que serão vistas mais adiante no seu curso).

De qualquer modo, essa técnica de levantar perguntas, ou hipóteses sobre os dados e buscar respondê-las por análises quantitativas e a a visualização dos dados é bastante útil e aplicaremos um pouco dessa técnica aqui.

### Qual o país com mais vinhos avaliados e qual a posição do Brasil?

Aqui é útil empregarmos a função countplot do Seaborn, ou empregar o groupby do Pandas para obter essas respostas.

plt.figure(figsize=(14,4))  
sns.countplot(x=wine.country)  
plt.title('Country Wines')  
plt.xticks(rotation=45)  
plt.show()



pd.DataFrame(wine.country.value\_counts()).reset\_index()[23:30]

index country  
23 Croatia 73  
24 Mexico 70  
25 Moldova 59  
26 Brazil 52  
27 Lebanon 35  
28 Morocco 28  
29 Peru 16

### Qual a origem e o tipo de vinhos mais baratos e mais caros avaliados?

Os operadores nlargest e nsmallest do Pandas são bastante úteis para responder esse tipo de questões sobre os dados.

wine.loc[wine.price.nlargest(5).index][['country','title','variety','price']]

country ... price  
80290 France ... 3300.0  
15840 France ... 2500.0  
98380 France ... 2500.0  
120391 US ... 2013.0  
65352 France ... 2000.0  
  
[5 rows x 4 columns]

wine.loc[wine.price.nsmallest(5).index][['country','title','variety','price']]

country ... price  
1987 Spain ... 4.0  
20484 US ... 4.0  
29553 Argentina ... 4.0  
31530 US ... 4.0  
59507 US ... 4.0  
  
[5 rows x 4 columns]

E não deve ser surpresa para você que os vinhos Bordeaux e Franceses estão entre os mais caros vinhos avaliados.

### Como estão distribuídos os preços dos vinhos no Brasil e no Mundo?

As análises gráficas muitas vezes requerem sucessivos refinamentos.

plt.figure(figsize=(12,4))  
sns.kdeplot(wine[wine.country == 'Brazil'].price,label='Brazil')  
sns.kdeplot(wine.price,label='World')  
plt.title('Price Distributions')  
plt.legend()  
plt.show()



Ao fazermos o gráfico de distribuição acima podemos ver que a maior parte dos vinhos (área abaixo da curva) se encontra abaixo dos USD 100. Mas a visualização é bastante prejudicada pela dispersão dos valores e os valores discrepantes, como podemos ver com um gráfico boxplot.

plt.figure(figsize=(12,4))  
plt.boxplot(wine[wine.country == 'Brazil'].price,positions=[1])  
plt.boxplot(wine.price,positions=[2])  
plt.title('Price Distributions')  
plt.xticks(ticks=[1,2], labels=['Brazil','World'])  
plt.show()



Mas o gráfico acima não exibe os valores! De fato, como existem valores nulos em price o Matplotlib falha em calcular as estatísticas dos quartis requeridas. Desse modo temos que excluir os valores nulos. Há muitas formas de se fazer isso e adotaremos aqui o modo mais simples e direto.

wine\_not\_null = wine[ ~ np.isnan( wine.price ) ]   
  
plt.figure(figsize=(12,4))  
plt.boxplot(wine\_not\_null[wine\_not\_null.country == 'Brazil'].price,positions=[1])  
plt.boxplot(wine\_not\_null.price,positions=[2])  
plt.title('Price Distributions')  
plt.xticks(ticks=[1,2], labels=['Brazil','World'])  
plt.show()



Fica evidente assim a presença de inúmeros outliers nos dados e podemos, assim, excluí-los para uma melhor visualização.

plt.figure(figsize=(12,4))  
plt.boxplot(wine\_not\_null[wine\_not\_null.country == 'Brazil'].price,positions=[1],showfliers=False)  
plt.boxplot(wine\_not\_null.price,positions=[2],showfliers=False)  
plt.title('Price Distributions')  
plt.xticks(ticks=[1,2], labels=['Brazil','World'])  
plt.show()



Observamos assim que vinhos acima de USD 80 são valores discrepantes no mercado, estando a maior parte dos vinhos abaixo de USD 50-40, e que os valores dos vinhos nacionais não são, afinal, tão distantes do mercado mundial.

### De que regiões vem os vinhos Brasileiros avaliados?

Talvez você esteja curioso em saber a origem dos vinhos nacionais e essa é uma pergunta bastante simples de ser respondida, e não deve ser surpresa descobrir que a maior parte dos vinhos nacionais avaliados vem da Serra Gaúcha.

wine[wine.country == 'Brazil'].province.unique()

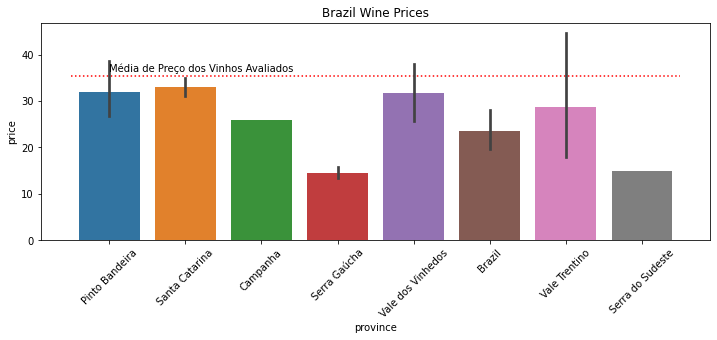
array(['Pinto Bandeira', 'Santa Catarina', 'Campanha', 'Serra Gaúcha',  
 'Vale dos Vinhedos', 'Brazil', 'Vale Trentino', 'Serra do Sudeste'],  
 dtype=object)

wine[wine.country == 'Brazil'].groupby('province').province.count()

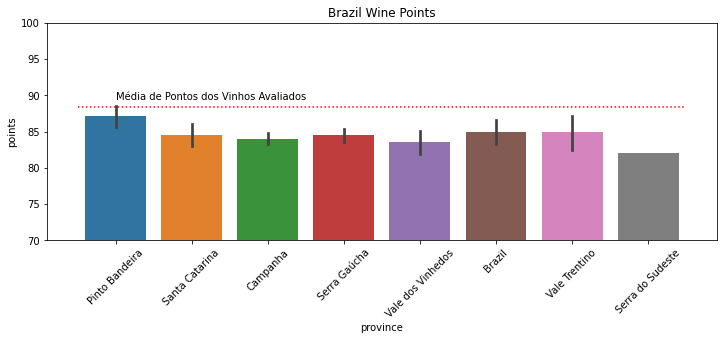
province  
Brazil 7  
Campanha 4  
Pinto Bandeira 6  
Santa Catarina 2  
Serra Gaúcha 18  
Serra do Sudeste 1  
Vale Trentino 5  
Vale dos Vinhedos 9  
Name: province, dtype: int64

Eles, entretanto, não são os vinhos em média mais caros do Brasil.

plt.figure(figsize=(12,4))  
sns.barplot(x=wine[wine.country == 'Brazil'].province, y=wine[wine.country == 'Brazil'].price)  
plt.title('Brazil Wine Prices')  
plt.hlines(wine.price.mean(),-0.5,7.5,color='r',linestyles=':')  
plt.text(0, wine.price.mean()+1,'Média de Preço dos Vinhos Avaliados')  
plt.xticks(rotation=45)  
plt.show()



plt.figure(figsize=(12,4))  
sns.barplot(x=wine[wine.country == 'Brazil'].province, y=wine[wine.country == 'Brazil'].points)  
plt.title('Brazil Wine Points')  
plt.hlines(wine.points.mean(),-0.5,7.5,color='r',linestyles=':')  
plt.text(0, wine.points.mean()+1,'Média de Pontos dos Vinhos Avaliados')  
plt.xticks(rotation=45)  
plt.ylim([70,100])  
plt.show()



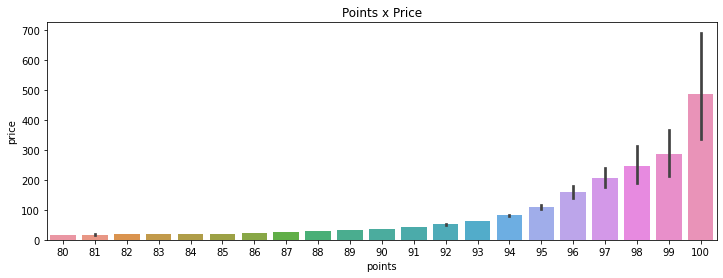
E, portanto, talvez você não precise pagar mais para tomar um bom vinho.

Mas de qualquer modo, os vinhos brasileiros estão têm preços abaixo da média mundial, mas também avaliações abaixo da média.

### Os melhores vinhos são os vinhos mais caros?

É claro que aqui consideramos *melhores* os vinhos melhor avaliados. De fato, uma simples exibição dos dados permite ver essa relação.

plt.figure(figsize=(12,4))  
sns.barplot(x=wine.points, y=wine.price)  
plt.title('Points x Price')  
plt.show()

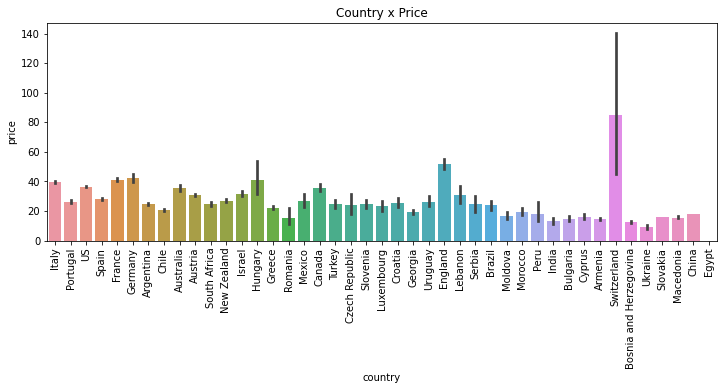


Mais adiante no curso, quando você conhecer técnicas estatísticas como testes de hipóteses, pode aplicar neste caso e confirmar a dependência e as avaliações e o preço, que observamos aqui visualmente.

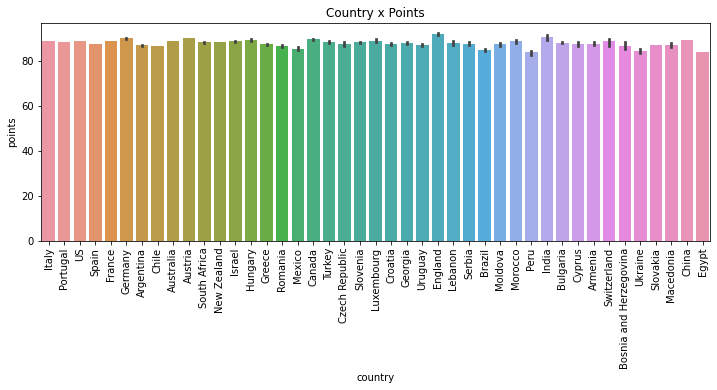
### Os vinhos Franceses são mesmo os mais caros e bem avaliados?

Embora os vinhos Franceses superem em preço e pontuação as médias globais, uma análise mais detalhada mostra que existem muitos países que parecem apresentar vinhos melhores e mais caros.

plt.figure(figsize=(12,4))  
sns.barplot(x=wine.country, y=wine.price)  
plt.title('Country x Price')  
plt.xticks(rotation=90)  
plt.show()

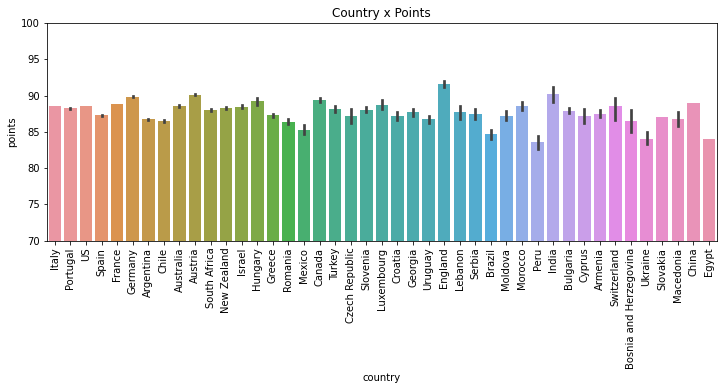


plt.figure(figsize=(12,4))  
sns.barplot(x=wine.country, y=wine.points)  
plt.title('Country x Points')  
plt.xticks(rotation=90)  
plt.show()



Novamente a visualização falha em mostrar adequadamente os dados, mas aqui o refinamento é bastante simples e basta reconfigurarmos os limites do eixo de pontos.

plt.figure(figsize=(12,4))  
sns.barplot(x='country', y='points', data=wine)  
plt.title('Country x Points')  
plt.xticks(rotation=90)  
plt.ylim([70,100])  
plt.show()



### Conclusão das Análises

Podemos nos estender por muito mais análises desses dados, mas quando trabalhamos em um caso real o objetivo da análise vai direcionar e limitar essas possibilidades.

De qualquer modo qualquer análise deve ter uma conclusão e aqui poderíamos resumir alguns dos nossos resultados.

Os vinhos brasileiros avaliados são produzidos em sua maioria na região sul do país. São vinhos bastante baratos em comparação ao mercado internacional, mas também vinhos mal avaliados quando comparados à média mundial. O preço e pontuação dos vinhos são bastante relacionados, mas há uma faixa de vinhos bastante bem avaliados e com preços abaixo de USD 50 (gráfico abaixo), o que sugere que não é necessário gastar mais do que isso para beber um *excelente* vinho. Os vinhos Franceses, bastante famosos, embora superem em qualidade e preço a média dos demais países, são surpreendentemente superados por outros países como a Índia, Inglaterra e Alemanha.

Novamente, essas análises são apenas um exercício sobre as técnicas que queremos explorar e conclusões mais definitivas deveríamos deixar a cargo de *sommeliers*.

# Caso: Gapminder, *Correlações*

Gapminder, <https://www.gapminder.org/data/>, é uma importante fonte de dados aberta que contêm diversas informações e índices relacionados ao desenvolvimento dos países.

Aqui diversos dados já foram coletados e agregados e estaremos interessados em explorar unicamente algumas *correlações* dos dados. Correlações estão entre as primeiras e mais importantes análises bivariadas (relação entre duas variáveis) que devemos fazer sobre os dados e neste ponto você pode achar útil voltar na aula 2 quando estudamos covariância e correlação.

df = pd.read\_csv('https://raw.githubusercontent.com/Rogerio-mack/Visualizacao-de-Dados-em-Python/main/data/gapminder\_2015.csv')  
df['cod\_continent'] = df[['continent']].astype('category').apply(lambda x: x.cat.codes)  
df.head()

continent country year ... CPI Population cod\_continent  
0 Africa Botswana 2015 ... 63.0 2120000 0  
1 Africa Burkina Faso 2015 ... 38.0 18100000 0  
2 Africa Cote d'Ivoire 2015 ... 32.0 23200000 0  
3 Africa Egypt 2015 ... 36.0 92400000 0  
4 Africa Kenya 2015 ... 25.0 47900000 0  
  
[5 rows x 17 columns]

## Correlação

A covariância busca mostrar se há um comportamento de **interdependência linear entre duas variáveis** o quê, como você já viu, não elimina outros tipos de interdependência e não significa necessariamente uma relações de causa efeito.

A covariância de uma amostra é definida como:

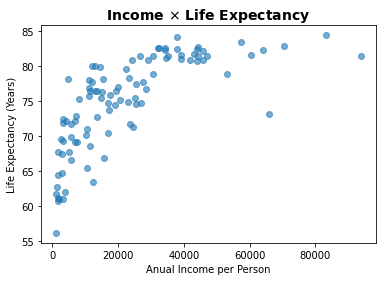
e, da mesma forma, a covariância populacional é dada por:

A covariância, entretanto, é uma medida dimensional e, portanto, afetada pelas unidades de medida empregadas nos conjuntos de dados e . Para corrigir isso podemos então empregar o coeficiente de correlação, que nada mais é que covariância normalizada (e portanto uma medida adimensional):

O **coeficiente de determinação**, ou mais exatamente o **coeficiente de determinação de Pearson** pode ser então calculado como:

Uma forma de observamos interdependências lineares dos dados é exibirmos um gráficos de pontos ou de dispersão das duas variáveis.

plt.scatter(df['income\_per\_person'],df['LifeExpect'] , alpha=0.6)  
  
plt.title('Income $\\times$ Life Expectancy', fontsize=14, weight='bold')  
plt.xlabel('Anual Income per Person')  
plt.ylabel('Life Expectancy (Years)')  
  
plt.show()



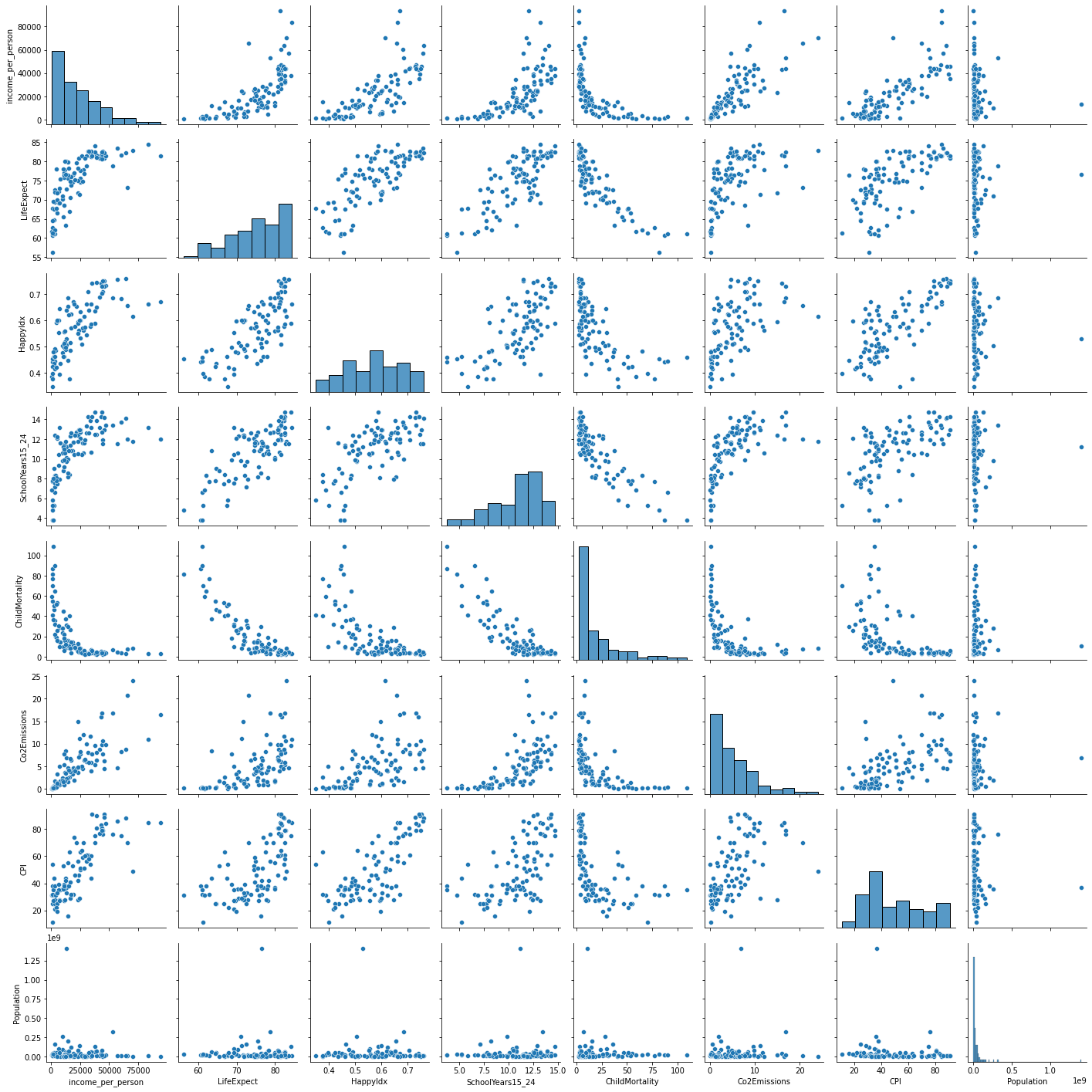
## Correlogramas

Correlogramas nada mais são do que vários gráficos de pontos de diferentes pares de variáveis. Desse modo, você explora as relações dos dados analisando separadamente cada par de variáveis. É uma forma de evitarmos o *overplotting* e dividirmos o problema da análise de relações em várias partes.

Podemos fazer isso com o Matplotlib, mas havendo várias variáveis pode ser necessário fazer um laço de programa para produzir os diferentes gráficos de dispersão de cada par de variável (em geral empregamos somente as quantitativas, numéricas) e logo a seguir você verá um laço de programa como esse. Mas essa tarefa pode ser um tanto tediosa.

O Seaborn fornece uma função para exibir, em um único comando, os diferentes pares de variáveis e é, geralmente, a forma mais simples e empregada. Apenas para fins de apresentação no texto limitamos a quantidade de pares de variáveis a serem exploradas. De qualquer modo, embora possível, é recomendável evitar execuções do tipo sns.pairs(df) que exibirão *todas* as relações. Elas levarão muito tempo para serem processadas e, em muitos casos, produzem gráficos desnecessários (variáveis categóricas também serão incluídas neste caso).

sns.pairplot(df[['income\_per\_person', 'LifeExpect', 'HappyIdx','SchoolYears15\_24', 'ChildMortality', 'Co2Emissions',  
 'CPI', 'Population']])  
plt.show()



Observando o gráfico acima algumas relações se destacam por apresentar uma relação mais linear como LifeExpect ChildMortality e SchoolYears15\_24 ChildMortality.

De fato, essas são relações que apresentam um coeficiente de correlação bastante alto e podemos identificar essas correlações produzindo uma matriz de correlação desses pares de variáveis e identificando seus maiores valores. Fazemos isso a seguir.

pd.options.display.max\_rows = 999  
df[['income\_per\_person',  
 'LifeExpect', 'HappyIdx',  
 'SchoolYears15\_24', 'ChildMortality', 'Co2Emissions',  
 'CPI', 'Population']].corr()

income\_per\_person LifeExpect ... CPI Population  
income\_per\_person 1.000000 0.710835 ... 0.805391 -0.052656  
LifeExpect 0.710835 1.000000 ... 0.651758 0.017118  
HappyIdx 0.727471 0.750452 ... 0.729569 -0.030911  
SchoolYears15\_24 0.652686 0.772257 ... 0.583404 0.023106  
ChildMortality -0.600019 -0.878321 ... -0.527718 -0.039804  
Co2Emissions 0.796020 0.532751 ... 0.558390 0.059660  
CPI 0.805391 0.651758 ... 1.000000 -0.086626  
Population -0.052656 0.017118 ... -0.086626 1.000000  
  
[8 rows x 8 columns]

df[['income\_per\_person',  
 'LifeExpect', 'HappyIdx',  
 'SchoolYears15\_24', 'ChildMortality', 'Co2Emissions',  
 'CPI', 'Population']].corr().abs().unstack().drop\_duplicates().sort\_values(ascending=False).nlargest(5)

income\_per\_person income\_per\_person 1.000000  
LifeExpect ChildMortality 0.878321  
SchoolYears15\_24 ChildMortality 0.851195  
income\_per\_person CPI 0.805391  
 Co2Emissions 0.796020  
dtype: float64

## O mesmo em R

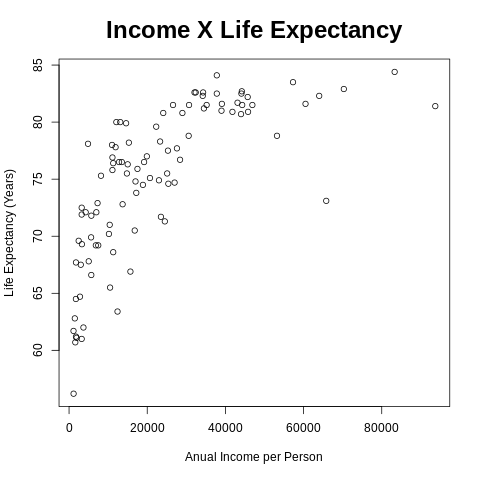
Os mesmos resultados podem ser igualmente obtidos em R. A função pairs, por exemplo, corresponde ao correlograma.

%load\_ext rpy2.ipython

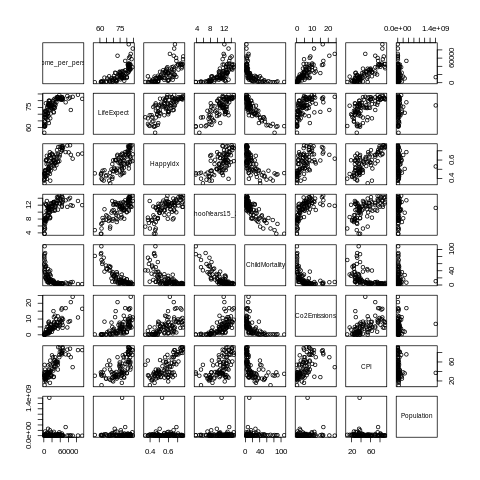
%%R  
df = read.csv('https://raw.githubusercontent.com/Rogerio-mack/Visualizacao-de-Dados-em-Python/main/data/gapminder\_2015.csv')  
head(df)

continent country year demox\_eiu income\_per\_person invest\_.\_gdp  
1 Africa Botswana 2015 78.7 15700 32.1  
2 Africa Burkina Faso 2015 47.0 1600 24.3  
3 Africa Cote d'Ivoire 2015 33.1 3230 20.1  
4 Africa Egypt 2015 31.8 10200 14.3  
5 Africa Kenya 2015 53.3 2800 21.5  
6 Africa Malawi 2015 55.5 1140 12.2  
 tax\_.\_gdp gini\_index LifeExpect HappyIdx SchoolYears15\_24 VacineBelieve  
1 24.7 60.5 66.9 0.376 8.40 NA  
2 15.1 35.5 60.7 0.442 3.76 NA  
3 15.4 41.6 61.0 0.445 6.59 NA  
4 12.5 31.2 70.2 0.476 10.60 NA  
5 16.3 41.5 64.7 0.436 9.06 NA  
6 15.2 45.5 61.7 0.387 6.85 NA  
 ChildMortality Co2Emissions CPI Population  
1 40.7 2.5600 63 2120000  
2 86.8 0.1820 38 18100000  
3 90.0 0.4050 32 23200000  
4 23.6 2.3700 36 92400000  
5 46.3 0.3410 25 47900000  
6 59.2 0.0669 31 16700000

%%R  
  
plot(df[, c('income\_per\_person')],df[ , c('LifeExpect')],   
 main='Income X Life Expectancy',  
 xlab='Anual Income per Person',  
 ylab='Life Expectancy (Years)',  
 cex.main=2)



%%R  
  
pairs(df[, c('income\_per\_person', 'LifeExpect', 'HappyIdx','SchoolYears15\_24', 'ChildMortality', 'Co2Emissions',  
 'CPI', 'Population')])



R e Python podem diferir algumas vezes no tratamento dos dados. O Python retorna o cálculo de médias e outras estatísticas mesmo na presença de valores ausentes. Já o R não tem o mesmo comportamento. Abaixo, para obtermos os valores de correlação para o atributos HappyIdx que possui alguns valores, aplicamos a técnica de imputar os valores médios nos valores ausentes.

%%R  
sapply(df[ , c('HappyIdx')], is.na)  
# any(sapply(df[ , c('HappyIdx')], is.na))  
  
# cuidado... NA <> null in R  
# sapply(df[ , c('HappyIdx')], is.na)  
# any(sapply(df[ , c('HappyIdx')], is.na))

[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE FALSE  
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
[49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[97] FALSE FALSE

%%R  
df[ is.na(df[ , c('HappyIdx')]), ]$HappyIdx = mean(df[ , c('HappyIdx')], na.rm=T)

%%R  
any(sapply(df[ , c('HappyIdx')], is.na))

[1] FALSE

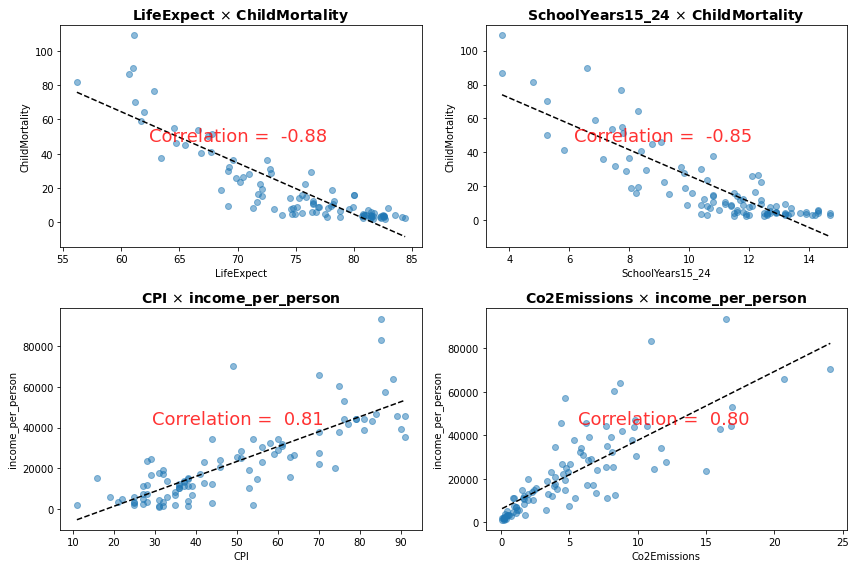
%%R  
  
cor(df[, c('income\_per\_person',  
 'LifeExpect', 'HappyIdx',  
 'SchoolYears15\_24', 'ChildMortality', 'Co2Emissions',  
 'CPI', 'Population')])

income\_per\_person LifeExpect HappyIdx SchoolYears15\_24  
income\_per\_person 1.00000000 0.71083478 0.72280345 0.65268626  
LifeExpect 0.71083478 1.00000000 0.73632393 0.77225731  
HappyIdx 0.72280345 0.73632393 1.00000000 0.65335710  
SchoolYears15\_24 0.65268626 0.77225731 0.65335710 1.00000000  
ChildMortality -0.60001883 -0.87832145 -0.64427042 -0.85119475  
Co2Emissions 0.79601998 0.53275092 0.55465444 0.62886628  
CPI 0.80539119 0.65175832 0.72438664 0.58340363  
Population -0.05265571 0.01711828 -0.03086819 0.02310579  
 ChildMortality Co2Emissions CPI Population  
income\_per\_person -0.60001883 0.79601998 0.80539119 -0.05265571  
LifeExpect -0.87832145 0.53275092 0.65175832 0.01711828  
HappyIdx -0.64427042 0.55465444 0.72438664 -0.03086819  
SchoolYears15\_24 -0.85119475 0.62886628 0.58340363 0.02310579  
ChildMortality 1.00000000 -0.54285070 -0.52771807 -0.03980397  
Co2Emissions -0.54285070 1.00000000 0.55838957 0.05966049  
CPI -0.52771807 0.55838957 1.00000000 -0.08662597  
Population -0.03980397 0.05966049 -0.08662597 1.00000000

Com R ou Python, o ponto é que com esses maiores valores de correlação, ou pelo correlograma, identificamos os pares de variáveis de interesse, isto é, variáveis que mais apresentam uma interdependência linear, e podemos aprofundar nossa análise, por exemplo, produzindo linhas de tendência para elas.

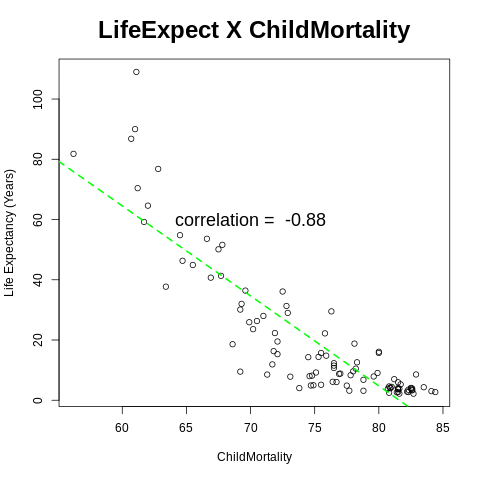
Implementamos isso abaixo com um laço de programa de gráficos de pontos do Matplotlib, e algo semelhante pode ser facilmente produzido com o Seaborn.

plt.figure(figsize=(12, 8))   
  
def linear(x, a, b):  
 return a\*x + b  
  
for i, pairs in enumerate( [ ['LifeExpect','ChildMortality',],  
 ['SchoolYears15\_24','ChildMortality'],  
 ['CPI','income\_per\_person'],  
 ['Co2Emissions','income\_per\_person'] ] ):   
   
 coefs, covar = curve\_fit(linear, df[pairs[0]], df[pairs[1]])  
 x = np.linspace(df[pairs[0]].min(), df[pairs[0]].max(), 100)  
 y = linear(x,\*coefs)  
  
 plt.subplot(2,2,i+1)  
 plt.scatter(df[pairs[0]], df[pairs[1]], alpha=0.5)  
  
 plt.plot(x,y,color='black', linestyle='dashed', lw=1.5)  
  
 texto = f'Correlation = {df[[pairs[0],pairs[1]]].corr().unstack()[1]:.2f} '  
 plt.text(( plt.gca().get\_xlim()[1] + plt.gca().get\_xlim()[0])/2,  
 ( plt.gca().get\_ylim()[1] + plt.gca().get\_ylim()[0])/2,  
 texto, ha='center', va='center', fontsize=18, color='red', alpha=0.8)  
  
 plt.title(pairs[0] + ' $\\times$ ' + pairs[1], fontsize=14, weight='bold')  
 plt.xlabel(pairs[0])  
 plt.ylabel(pairs[1])  
  
plt.tight\_layout()  
plt.show()



As mesmas linhas de tendência também podem ser feitas em R como abaixo para ChildMortality e LifeExpect.

%%R  
  
plot(df[, c('LifeExpect')],df[ , c('ChildMortality')],   
 main='LifeExpect X ChildMortality',  
 xlab='ChildMortality',  
 ylab='Life Expectancy (Years)',  
 cex.main=2)  
  
abline(lm(ChildMortality ~ LifeExpect, data=df),col='green',lty=2, lwd=2)  
text(70,60, paste('correlation = ' , round( cor( df[, c('LifeExpect')],df[ , c('ChildMortality')] ), 2)), cex=1.5 )



Você pode ter a impressão que isso é muito mais fácil em R que Pythom, mas pesquise a função sns.lmplot() do pacote Seaborn, você verá que é também bastante simples fazer isso com Python.

É importante notar que, embora tenhamos explorado relações lineares dos dados, o correlograma não se limita a apresentar somente relações lineares e poderia ajudar a identificar muitos outros padrões de relacionamento entre os dados.

## Mapas de Calor

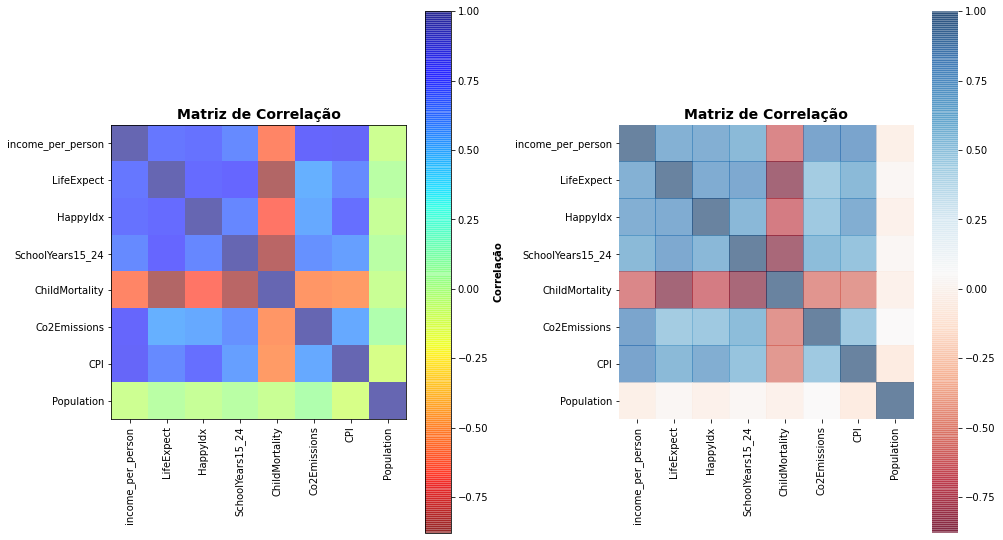
Mapas de Calor ou *Heat Maps* permitem visualizar dados multivariados exibindo variáveis ​​discretas (categóricas ou numéricas) nas linhas e colunas de um conjunto retangular de células e uma dimensão quantitativa colorindo esses elementos. Os mapas de calor são ótimas representações para exibir variações de valores em várias variáveis e identificar grupos de dados que apresentam um mesmo padrão de valores.

### Mapa de Correlação

Um caso especial muito comum de uso dos Mapas de Calor é para exibirmos uma matriz de correlação. Esse gráfico pode substituir o Correlograma para a identificação de correlações, isto é, relações lineares. Mas você deve lembrar que podem haver outras relações e, neste caso, o Correlograma é um instrumento de análise mais geral.

O Seaborn já tem uma função específica para mapas de calor e a função heatmap espera receber um narray, ou um DataFrame, empregando neste caso as informações de índice e coluna para rotular o gráfico.

plt.figure(figsize=(14,8))  
  
matriz = df[['income\_per\_person',  
 'LifeExpect', 'HappyIdx',  
 'SchoolYears15\_24', 'ChildMortality', 'Co2Emissions',  
 'CPI', 'Population']].corr()  
  
plt.subplot(1,2,1)  
plt.imshow(matriz, cmap='jet\_r', alpha=0.6)  
  
plt.xticks(np.arange(len( matriz.columns)), matriz.columns.to\_list(), rotation=90)  
plt.yticks(np.arange(len( matriz.columns)), matriz.columns.to\_list())  
  
cbar = plt.colorbar()  
cbar.set\_label('$\\bf{Correlação}$')  
  
plt.title('Matriz de Correlação', fontsize=14, weight='bold')  
  
plt.subplot(1,2,2)  
sns.heatmap(matriz, cmap='RdBu', square=True, alpha=0.6)   
  
plt.title('Matriz de Correlação', fontsize=14, weight='bold')  
  
plt.tight\_layout()  
plt.show()



Em quaisquer dos gráficos acima você pode indentificar, por exemplo, a alta correlação de ChildMortality com diversos outros indicadores, do mesmo modo que observamos antes no Correlograma.

# Caso: Dados de Remunerações, *Tabelas de Contingência*

Neste último caso vamos analisar dados de Remunerações do pacote ISLR do R empregando tabelas de contingência.

## Tabelas de Contingência

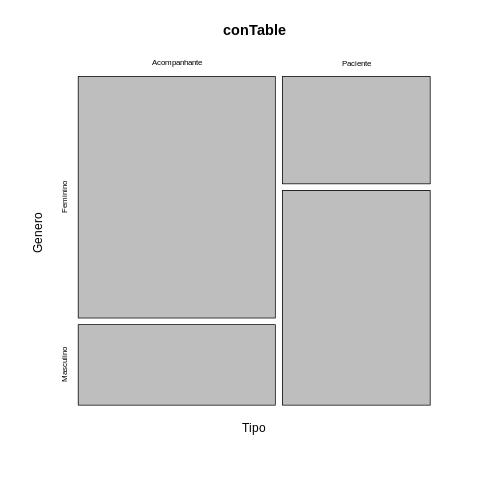
Uma tabela de contingência é bastante útil para sumarizarmos grandes conjunto de dados. A ideia de uma tabela de contingência é fornecer a frequencia de certos valores de forma cruzada, isto é, com relação aos valores de outros atributos. A análise anterior, de correlações, é basicamente uma análise entre duas variáveis numéricas e não permite uma análise de interdependência entre variáveis categóricas, ou categóricas e numéricas. Assim, as tabelas de contingência, são um instrumento importante quando queremos analisar as relações entre variáveis não numéricas.

O exemplo mais simples de tabela de contingência em R pode ser dado pela aplicação direta do comando table() (para Python/Pandas ver o comando crosstab()).

%%R  
df = data.frame(  
 "Tipo" = c("Acompanhante", "Paciente", "Paciente", "Acompanhante", "Acompanhante", "Acompanhante", "Paciente"),  
 "Genero" = c("Feminino", "Masculino", "Masculino", "Feminino", "Masculino", "Feminino", "Feminino")  
)  
   
conTable = table(df)  
print(conTable)

Genero  
Tipo Feminino Masculino  
 Acompanhante 3 1  
 Paciente 1 2

%%R  
mosaicplot(conTable)



Tanto o gráfico como os valores da tabela permitem observar aqui a quantidades de cada par (tipo, genero), e portanto, as frequencias relativas ou probabilidades desses valores.

## Dados de Remunerações

O conjunto de dados Wage do pacote ISLR traz uma série de informações sobre remunerações.

%%R  
install.packages('ISLR')

%%R  
library(ISLR)  
  
head(Wage)

year age maritl race education region  
231655 2006 18 1. Never Married 1. White 1. < HS Grad 2. Middle Atlantic  
86582 2004 24 1. Never Married 1. White 4. College Grad 2. Middle Atlantic  
161300 2003 45 2. Married 1. White 3. Some College 2. Middle Atlantic  
155159 2003 43 2. Married 3. Asian 4. College Grad 2. Middle Atlantic  
11443 2005 50 4. Divorced 1. White 2. HS Grad 2. Middle Atlantic  
376662 2008 54 2. Married 1. White 4. College Grad 2. Middle Atlantic  
 jobclass health health\_ins logwage wage wage\_cat  
231655 1. Industrial 1. <=Good 2. No 4.318063 75.04315 Below  
86582 2. Information 2. >=Very Good 2. No 4.255273 70.47602 Below  
161300 1. Industrial 1. <=Good 1. Yes 4.875061 130.98218 Above  
155159 2. Information 2. >=Very Good 1. Yes 5.041393 154.68529 Above  
11443 2. Information 1. <=Good 1. Yes 4.318063 75.04315 Below  
376662 2. Information 2. >=Very Good 1. Yes 4.845098 127.11574 Above

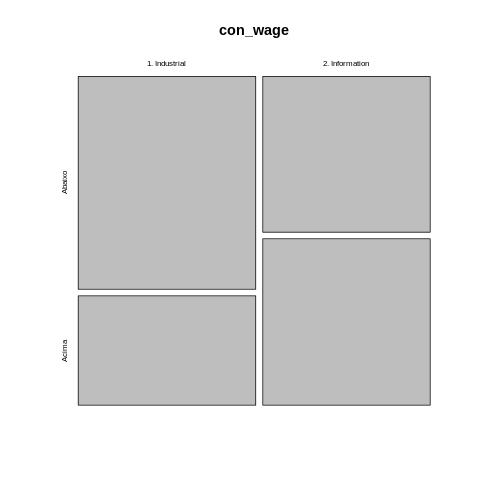
### Criando dados Derivados

Uma técnica bastante útil na análise de dados é a criação de dados derivados. Dados derivados podem vir de fontes externas, como empregar o Google Maps API para obter a longitude e latitude ou a distância de duas cidades cujos nomes fazem parte da sua base de dados. Podem incluir conversões como alterar valores de Real para Dólar, empregando para isso os valores de correção do dia a que se referem os dados, ou ainda obter o dia da semana ou se uma data é feriado ou não, o que pode ser importante para um sistema que analisa um movimento de vendas.

Também podemos criar atributos derivados discretizando valores numéricos e no nosso exemplo aqui vamos criar uma variável simples para identificar valores acima e abaixo da média das remunerações, sendo a variável categórica em que vamos concentrar a nossa análise.

%%R  
Wage$wage\_cat = as.factor(ifelse(Wage$wage > mean(Wage$wage), "Acima", "Abaixo"))  
   
con\_wage = table(Wage$jobclass,Wage$wage\_cat)  
  
print(con\_wage)  
mosaicplot(con\_wage)

Abaixo Acima  
 1. Industrial 1020 524  
 2. Information 704 752



Somente com isso já podemos notar que a área indutrial traz uma maior parcela de remunerações abaixo da média, enquanto o setor de informação apresenta as remunerações mais igualmente distribuídas e isso pode fornecer importantes *insights* para quem toma decisões sobre as remunerações!

Além das quantidades absolutas você pode estar interessado em obter os valores das proporções, isto é as frequencias relativas ou as probabilidades desses valores. O comando prop.table() pode então ser empregado neste caso proporcionando as frequencias marginais total ou por cada um dos valores envolvidos.

%%R  
# total  
prop.table(con\_wage)

Abaixo Acima  
 1. Industrial 0.3400000 0.1746667  
 2. Information 0.2346667 0.2506667

O valor 0.34 mostra que 34% dos trabalhadores corresponde a categoria Industrial com Remunerações abaixo da Média.

%%R  
# por linha  
prop.table(con\_wage, margin = 1)

Abaixo Acima  
 1. Industrial 0.6606218 0.3393782  
 2. Information 0.4835165 0.5164835

O valor 0.66 mostra que 66% dos trabalhadores da categoria Industrial recebem Remunerações abaixo da Média.

%%R  
# por coluna  
prop.table(con\_wage, margin = 2)

Abaixo Acima  
 1. Industrial 0.5916473 0.4106583  
 2. Information 0.4083527 0.5893417

O valor 0.59 mostra que 59% dos trabalhadores com Remunerações abaixo da Média são do Setor Industrial.

A somatária de linhas de colunas pode ser obtida com o comando addmargins() o que pode ser útil em casos onde trabalhamos com um grande número de variáveis.

%%R  
addmargins(con\_wage)

Abaixo Acima Sum  
 1. Industrial 1020 524 1544  
 2. Information 704 752 1456  
 Sum 1724 1276 3000

%%R  
addmargins(prop.table(con\_wage))

Abaixo Acima Sum  
 1. Industrial 0.3400000 0.1746667 0.5146667  
 2. Information 0.2346667 0.2506667 0.4853333  
 Sum 0.5746667 0.4253333 1.0000000

%%R  
addmargins(prop.table(con\_wage, margin=1))

Abaixo Acima Sum  
 1. Industrial 0.6606218 0.3393782 1.0000000  
 2. Information 0.4835165 0.5164835 1.0000000  
 Sum 1.1441382 0.8558618 2.0000000

%%R  
addmargins(prop.table(con\_wage, margin=2))

Abaixo Acima Sum  
 1. Industrial 0.5916473 0.4106583 1.0023056  
 2. Information 0.4083527 0.5893417 0.9976944  
 Sum 1.0000000 1.0000000 2.0000000

Obviamente com o parâmetro margin é necessário cuidado com a interpretação desses valores.

## Interdependência

Mas como verificar se há de fato uma interdependência, uma relação verdadeira, entre duas variáveis categóricas a exemplo do que fizemos com variáveis numéricas ao explorar as correlações?

A resposta a essa pergunta exige o conhecimento de testes de hipóteses que você verá mais adiante no curso. Mas aqui, fazendo uma anologia ao que você já aprendeu sobre correlação, podemos fazer uma abordagem prática e aprender a empregar um teste de hipótese simples sem, entretanto, entrarmos nos aspectos mais conceituais que envolvem essa técnica.

Do mesmo modo que obtivemos o coeficiente de correlação como uma 'nota' para o um comportamento linear dos dados (sendo 1 a 'nota' de comportamento linear perfeito e 0 a de um comportamento nada linear), podemos pensar no teste de hipótese de Chi-Quadrado (existem muitos outros testes, como por exemplo o teste de Fisher, mas aqui empregaremos apenas um bastante simples) como um cálculo que irá nos retornar uma 'nota' para a existência ou não de uma interdependência entre duas variáveis categóricas.

%%R   
chisq.test(con\_wage)

Pearson's Chi-squared test with Yates' continuity correction  
  
data: con\_wage  
X-squared = 95.439, df = 1, p-value < 2.2e-16

De modo simples, somente se o p-value obtido acima for menor que 0.05 diremos que há uma dependência entre o setor de trabalho e a remuneração abaixo da média das remunerações com significância de 95%.

%%R  
chisq.test(con\_wage)$p.value

[1] 1.5251e-22

%%R  
if ( chisq.test(con\_wage)$p.value < 0.05 ){  
 print('Existe Interdependência entre as variáveis')  
} else {   
 print('Não Existe Interdependência entre as variáveis')  
}

[1] "Existe Interdependência entre as variáveis"

O teste confirma nossa impressão inicial sobre os dados e o fato de trabalhar ou não no setor industrial está bastante relacionado a obter-se uma remuneração abaixo ou acima da média. Nem sempre a observação dos dados permite chegarmos a uma conclusão clara e é nesse aspecto que os testes de hipótese são um importante instrumento de análise.

Testes semelhantes e mais poderosos que este você deve ver mais adiante no curso, mas isso já dá um bom instrumento de análise.

## Mais de 2 variáveis

Mas será que podemos acrescentar mais variáveis em nossa análise? Sim, as tabelas de contigência podem incluir outros níveis. Por exemplo, podemos incluir o item de raça se queremos verificar se as remunerações apresentam algum viés determinado pela raça dos indivíduos (e poderíamos aplicar o mesmo para genêro se tivéssemos essa informação na nossa base).

%%R  
con\_wage2 = xtabs( ~ jobclass + wage\_cat + race, data=Wage)  
   
ftable(con\_wage2)

race 1. White 2. Black 3. Asian 4. Other  
jobclass wage\_cat   
1. Industrial Abaixo 861 85 55 19  
 Acima 464 26 31 3  
2. Information Abaixo 539 126 30 9  
 Acima 616 56 74 6

O teste necessário aqui, entretanto, é um teste mais sofisticado, conhecido como Cochran-Mantel-Haenszel (CMH).

%%R  
test = mantelhaen.test(con\_wage2)  
  
print(test)  
  
if ( test$p.value < 0.05 ){  
 print('Existe Interdependência entre as variáveis')  
} else {   
 print('Não Existe Interdependência entre as variáveis')  
}

Mantel-Haenszel chi-squared test with continuity correction  
  
data: con\_wage2  
Mantel-Haenszel X-squared = 103.97, df = 1, p-value < 2.2e-16  
alternative hypothesis: true common odds ratio is not equal to 1  
95 percent confidence interval:  
 1.864255 2.512205  
sample estimates:  
common odds ratio   
 2.164114   
  
[1] "Existe Interdependência entre as variáveis"

O que quer dizer, no nosso caso, que nossa base demonstra que o setor de trabalho e a remuneração dos indivíduos é, no caso, dependente da raça dos indivíduos. Um mal sinal, :-(, mas é o que apontam os dados e só podemos torcer para esse tipo de análise levar a decisões que permitam corrigir esse viés.

Aplicando o mesmo teste para o atributo education também verificamos a dependência, mas com p-value bastante maior, o que indica uma dependência bastante menor que a raça.

%%R  
con\_wage3 = xtabs( ~ jobclass + wage\_cat + education, data=Wage)  
  
test = mantelhaen.test(con\_wage3)  
  
print(test)  
  
if ( test$p.value < 0.05 ){  
 print('Existe Interdependência entre as variáveis')  
} else {   
 print('Não Existe Interdependência entre as variáveis')  
}

Mantel-Haenszel chi-squared test with continuity correction  
  
data: con\_wage3  
Mantel-Haenszel X-squared = 6.6854, df = 1, p-value = 0.00972  
alternative hypothesis: true common odds ratio is not equal to 1  
95 percent confidence interval:  
 1.057345 1.475482  
sample estimates:  
common odds ratio   
 1.249037   
  
[1] "Existe Interdependência entre as variáveis"

# Para Saber Mais

1. Acesse <https://www.kaggle.com/> e pesquise por EDA. Você vai encontrar uma série de *cases* de exploração de dados que poderão interessar a você.
2. O próximo passo que você daria após essa análise dos dados talvez fosse a preparação dos dados tornando-os mais adequados para criação de modelos estatísticos ou de aprendizado de máquina. Muitas dessas transformações você verá adiante no curso, mas você pode já buscar conhecer algumas dessas técnicas. Para uma rápida introdução sobre isso em Python você pode acessar <https://www.kaggle.com/learn/data-cleaning>.
3. Quer ler mais sobre EDA? Acesse as referências online:

Hadley Wickham, Garrett Grolemund. **R for Data Science: Import, Tidy, Transform, Visualize, and Model Data**.

Disponível em: <https://r4ds.had.co.nz/>.

\_\_\_. **Engineering Statistics Handbook. Exploratory Data Analysis**. Disponível em: <https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm>

E leia sobre o processo de Análise Exploratória de Dados.

## Referências

1. Wickham, Hadley. **Tidy Data** (2013). Journal of Statistical Software. Disponível em: <https://www.jstatsoft.org/index.php/jss/article/view/v059i10/v59i10.pdf> Acesso: 17 de junho de 2021.
2. Hadley Wickham, Garrett Grolemund. **R for Data Science: Import, Tidy, Transform, Visualize, and Model Data**. O'Reilly Media Inc. (2017). (alternativamente disponível em: <https://r4ds.had.co.nz/>).
3. Jake VanderPlas. **Python Data Science Handbook**. O'Reilly Media, Inc. (2016). ISBN: 9781491912058. Disponível em: <https://jakevdp.github.io/PythonDataScienceHandbook/>. Acesso: 17.05.2021
4. Rogério de Oliveira. **Introdução à Visualização de Dados com Python**. Ed. Mackenzie. Coleção Conexão Inicial. *pre-print (2021)* Disponível em: <https://github.com/Rogerio-mack/Visualizacao-de-Dados-em-Python>. Acesso: 17.05.2021
5. Pipis, G. (2020) **Contingency Tables in R**. Disponível em: <https://predictivehacks.com/contingency-tables-in-r/> Acesso: 19.09.2021
6. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R. (2000). **CRISP-DM 1.0**. Disponível em: <https://the-modeling-agency.com/crisp-dm.pdf> Acesso em: 15 de Setembro de 2021.
7. \_\_\_. **Engineering Statistics Handbook. Exploratory Data Analysis**. Disponível em: <https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm> Acesso em: 15 de Setembro de 2021.