**Data Science**

**Curso I – Primeiros Passos:**

Link da documentação das bibliotecas que utilizaremos nesse curso:

- Pandas: <https://pandas.pydata.org/pandas-docs/stable/>;

- Seaborn: <http://seaborn.pydata.org/introduction.html>.

Vou colocar todo arquivo escrito na pasta da aula, mas aqui está o link do drive, de qualquer forma: <https://drive.google.com/drive/folders/1YV0fPKe-_f5beekL2cBUVSOsZ_2G6uaS>.

1. **Aula 1 – Data Science – Dados e Visualizações:**
   1. Podemos sim alterar o nome das colunas passando uma lista com os novos nomes a partir do atributo colunas do nosso df:

notas.columns = ['usuarioId', 'filmeId', 'nota', 'momento']

* 1. O pandas possui visualização e construção de dados a partir do método plot().
     1. O plot recebe o parâmetro kind=’’, onde podemos escolher o tipo de gráfico, como um hist de histograma, por exemplo. Se colocar somente plot() ele cria um plot normal onde não vemos nada:

notas.nota.plot(kind='hist')



notas.nota.plot()



* + - 1. O hist é um tipo de gráfico que conta uma história, mostrando no eixo x as notas que um filme recebeu e no y a quantidade de notas. Basicamente o inverso do plot.
    1. S.median(): retorna a mediana de uma series, mostrando exatamente qual o valor que dividi um conjunto de dados no meio:

notas.nota.median()



* + 1. Importamos a seaborn e fizemos um boxplot para visualizar melhor os gráficos:

import seaborn as sns

sns.boxplot(notas.nota)



* + - 1. A mediana é o traço no meio do box. Quartil 25 é o final esquerdo do box e 75 o final direito. As linhas com traço na ponta dos lados esquerdo e direito são, respectivamente os limites inferior e superior, ou seja, da mediana para as laterais do box temos 25% cada lado, os li e ls são outros 25%, somando 50% no box e 50% fora.
      2. Com esse gráfico podemos concluir que metade das pessoas, ou seja, o box inteiro, deram notas entre 3 e 4.
  1. O que aprendemos:
     1. Importar pandas que é uma biblioteca de análise de dados;
     2. Ler dados CSV;
     3. Fazer upload de arquivo para o Google Colab;
     4. Renomear colunas;
     5. Contar dados;
     6. Melhorar a visualização das informações.

1. **Aula 2 – Análise Exploratória:**
   1. Podemos usar queries para procurar coisas específicas nos nossos dados e selecionar a coluna que queremos, bem como a análise estatística depois dela:

notas.query('filmeId == 1').nota.mean()



* + 1. Dissemos que queremos a média das notas de todo filme que tenha o id == 1 dentro do nosso df de notas.
    2. Se quisermos tirar a média de todas as notas para cada id de filme teria que fazer isso infinitamente, mas, graças ao pandas, podemos agrupar todos os filmes pelo seu id com o df.groupby(‘coluna’) e tirar a média:

notas.groupby('filmeId').mean()



* + - 1. Contudo queremos só a coluna de notas, então podemos selecionar somente ela. O retorno será uma series. Para transformar em df basta colocar um to\_frame(). Como já vimos em outros cursos, podemos colocar tudo numa mesma linha para facilitar, ficando assim:

notas.groupby('filmeId').nota.mean().to\_frame()



* + 1. Após atribuir a uma variável, podemos fazer um histograma para ver a distribuição da média das notas:

medias\_por\_filme = notas.groupby('filmeId').nota.mean().to\_frame()

medias\_por\_filme.plot(kind='hist')



* + 1. Ou então pedir um sns.boxplot(medias\_por\_filme), porém, aprendi que ele não aceita frames, apenas series, nesse caso removi o s.to\_frame() da linha acima, rodei novamente e ele aceitou fazer o gráfico:

medias\_por\_filme = notas.groupby('filmeId').nota.mean()

sns.boxplot(medias\_por\_filme)



* + - 1. Geralmente boxplots são exibidos na vertical e não horizontal. Para alterar isso basta passar nosso s como y:

sns.boxplot(y=medias\_por\_filme)



* + 1. Para fazer um histograma no seaborn a nomenclatura muda, se tornando sns.distplot(), ou seja, gráfico de distribuição. Sua visualização é diferente também:

sns.distplot(medias\_por\_filme)



* + - 1. Podemos notar que esse histograma é diferente do que o pandas fez, mas isso ocorre de acordo com a distribuição de barras que cada um determina fazer.
      2. O pandas fez 10 separações/caixinhas/barras para distribuir as notas, já o seaborn fez MUITO mais, por isso a exibição fica diferente, mas isso não altera o resultado do gráfico, uma vez que os valores são um só.
      3. Entretanto, se quisermos podemos escolher quantas caixinhas o seaborn vai exibir, isto é, quantos bins=n queremos que ele faça para exibir o gráfico, passando esse como segundo parâmetro/argumento na hora de fazer o plot:

sns.distplot(medias\_por\_filme, bins=10)



* + - 1. Note que ao colocar 10 a distribuição fica exatamente igual ao do pandas.
      2. Precisamos tomar cuidado pois dependendo do número que escolhermos podemos passar diferentes mensagens com nosso gráfico, desse modo podendo passar uma mensagem errônea para frente ou até mesmo uma mentira. Precisamos ter cuidado com isso!
      3. Na prática o próprio seaborn já faz uns cálculos para determinar qual seria a quantidade de bins necessárias para cada gráfico.
      4. Além do hist. Ele também faz uma curva para mostrar mais suavemente como ficam essas notas de uma outra forma.
    1. Tanto o pandas quanto o seaborn utilizam uma biblioteca bem baixo nível para produzir esses gráficos. A biblioteca que ambos usam é a matplotlib.pyplot, sendo importada como plt. Biblioteca essa que também possui seu próprio histograma:

import matplotlib.pyplot as plt

plt.hist(medias\_por\_filme)



* + - 1. Essa lib possui diversos métodos, parâmetros que podem ser passados para alterar e modificar nossos gráficos de diversos modos que não vamos ver nesse curso, mas sim no data visualization mais pra frente.
    1. Voltando um pouco nas anotações vimos que o boxplot do seaborn ficou achatado quando mudamos para a horizontal. Para alterar isso podemos utilizar a biblioteca plt, uma vez que ela é quem está sendo rodada por baixo dos panos pelas outras libs que estamos utilizando. Para configurar o tamanho da imagem do gráfico utilizamos o plt.figure(figsize=(x,y)), podendo ser totalmente personalizado:

plt.figure(figsize=(5,8))

sns.boxplot(y=medias\_por\_filme)



* 1. Histograma e boxplot:
     1. O histograma também é conhecido como "distribuição de frequências", e é representado em um gráfico de colunas ou de barras a partir de um conjunto de dados previamente tabulado e dividido em classes (uniformes ou não uniformes);
     2. Já o boxplot, também conhecido como "diagrama de caixa", é um método alternativo para o histograma. Uma de suas particularidades é mostrar a faixa em que está concentrada a média de valores.
     3. Referências:
        1. <https://pt.wikipedia.org/wiki/Diagrama_de_caixa>;
        2. <https://pt.wikipedia.org/wiki/Histograma>.
  2. O que aprendemos:
     1. Trabalhar com query;
     2. O que são bins;
     3. Filtrar apenas uma coluna.

1. **Aula 3 – Variáveis:**
   1. Site para baixar arquivos com 5000 databases de filmes chamado the movie database (TMDB 5000): <https://www.kaggle.com/tmdb/tmdb-movie-metadata>.
      1. Virão 2 arquivos: um com os créditos, ou seja, as pessoas que participaram do filme e as informações sobre o filme em si.
   2. Categórica Nominal: variável de nome responsável por categorizar dados, como o caso de língua original de um filme. É um tipo de variável que não possui grandeza ou ordem, como as numéricas.
   3. Categórica Ordinal: ou seja, com ordem. Temos como exemplo a escolaridade, onde temos primeiro, segundo e terceiro grau por exemplo.
      1. Não possuí soma, ou seja, o primeiro mais o segundo não é igual ao 3°, por exemplo. No caso da escolaridade não é a mesma distância que o 1° grau tem do 2° que tem do 3°. É diferente das numéricas onde o 1 + 2 = 3.
      2. Não tem uma facilidade de realizar operações aritméticas.
   4. Quantitativo: variáveis de quantidade, como o nome diz. Temos como exemplo o orçamento/budget de filmes.
      1. Pode ser inclusive contínuo, ou seja, pode 0, pode ser 500 e pode ter infinitos valores no meio de ambos, como por exemplo, ter os números 15; 15,2; 15,37.... e assim por diante infinitamente.
   5. Quantitativo Intervalar: não possuí infinitos números decimais, por exemplo. É sempre determinado que vai de 1 em 1 ou de .5 em .5, como no caso de votos e notas de filmes, respectivamente.
   6. Para saber mais ordinais e nominais:
      1. As variáveis categóricas são aquelas que não possuem valores quantitativos, mas, ao contrário, são definidas por várias categorias, ou seja, representam uma classificação dos indivíduos. Podem ser nominais ou ordinais.
      2. Quando estamos trabalhando com categorias ordinais, existe uma ordenação entre as categorias. Por exemplo: escolaridade (primeiro, segundo e terceiro graus), estágios de uma doença (inicial, intermediário, terminal), mês de observação (janeiro, fevereiro, ..., dezembro), etc.
      3. Referência: <http://leg.ufpr.br/~silvia/CE055/node8.html>.
   7. O que aprendemos:
      1. Importar um CSV;
      2. Identificar o tipo de uma variável observando seu conteúdo;
      3. O que é categórica ordinal;
      4. O que é quantitativa.
2. **Aula 4 – Data Visualization:**
   1. Para visualizar categorias ordinais podemos utilizar o value\_counts() para ver quantas vezes uma língua aparece em um df de filmes, por exemplo, uma vez que o value\_counts() conta tudo, inclusive str’s:

tmdb['original\_language'].value\_counts()



* + 1. Para remover o index de uma series após converter ela para frame, usamos o método reset\_index():

tmdb['original\_language'].value\_counts().to\_frame().reset\_index()



* + - 1. Desse modo possuímos 2 colunas ao invés de 1 índice e 1 colunas, transformando assim uma série em um df.
      2. Note que os nomes das colunas estão bagunçados, podemos concertar facilmente como já sabemos:

contagem\_de\_lingua = tmdb['original\_language'].value\_counts().to\_frame().reset\_index()

contagem\_de\_lingua.columns = ['original\_language', 'total']

contagem\_de\_lingua.head()



* + 1. O seaborn possuí diversas formas de plotar categoricals. Podemos simplesmente procurar na documentação.
       1. Vamos usar o barplot(x, y, df):

sns.barplot(x = 'original\_language', y = 'total', data = contagem\_de\_lingua)



* + 1. Tivemos muito trabalho para chegar nesse gráfico, mas caso queiramos podemos utilizar o sns.catplot(x, kind=’’, data=df), que ao invés de fazer todo esse tratamento, podemos só pegar no df com todas as informações e pedir para que o seaborn faça um plot por categoria, passando quais são as categorias no x e dizendo qual o tipo de plot que queremos, nesse caso passando o ‘count’ como parâmetro para que ele conte quantas vezes cada categoria apareceu e depois fazendo as barras. No data passamos o df com todas as informações que importamos lá no começo.
       1. Podemos receber um erro se tentar rodar logo de cara dizendo que esse método não existe, porque, apesar dele existir, está apenas nas versões 0.9+ do seaborn e, ao verificar qual estamos rodando no colab com um print(), veremos que estamos rodando a 0.7. No meu caso ele está em uma versão recente:

print(sns.\_\_version\_\_)



* + - 1. Do mesmo modo que quando utilizando uma IDE local precisaríamos instalar a biblioteca, precisaremos instalar no notebook do colab, uma vez que ele está rodando em uma máquina virtual cloud e provavelmente não tem as versões mais recentes de todas as bibliotecas.
      2. Para instalar a versão desejada da lib, fazemos do mesmo modo que localmente, usando o !pip install lib. Se quiser ainda pode passar qual a versão da lib que quer instalar, fazendo a instalação ao rodar a célula:

!pip install seaborn=0.9.0

* + - 1. Para saber qual versão aquele método/função está disponível é bem fácil: CONSULTE A DOCUMENTAÇÃO!!!
      2. Resultado com o sns.catplot():

sns.catplot(x = 'original\_language', kind='count', data = tmdb)



* + - 1. Equivalente ao mesmo que fazendo toda a seleção de dados.
      2. A diferença é que ele não fica ordenado de acordo com a aparição como fizemos na seleção. Apenas demos sorte de en ter ficado em primeiro, mas ao ver os outros menores, podemos notar não está do maior para o menor.
      3. Para oficializar a versão, pós instalação, precisamos resetar o runtime.
  1. Gráfico de pizza quase nunca é recomendado em nenhuma situação.
     1. Vimos que no gráfico acima quase não conseguimos ver a diferença de cada língua, então decidimos fazer um gráfico de pizza/pie(en) para ver se melhora a visualização.
     2. O seaborn não possui esse tipo de gráfico, mas o matplotlib sim, então utilizamos ele passando 2 parâmetros, os valores e as labels no plt.pie(x, labels), e ele nos retorna esse resultado horrendo:

plt.pie(contagem\_de\_lingua['total'], labels = contagem\_de\_lingua['original\_language'])

plt.show()



* + 1. Por isso que quase nunca é recomendada sua utilização, pois fica visualmente horrível e muito mais confuso e pior do que estaria em qualquer outro.
    2. Considerando isso é bem importante sabermos qual é a história que queremos contar com nosso gráfico. Nesse caso nós queremos mostrar o quanto en aparece em relação aos outros idiomas, nesse caso, podemos separar os dados em 2 variáveis, uma com o total de inglês e outra com o total do resto. Para fazer isso podemos fazer uma contagem de valor da coluna de original\_language no nosso df com todos os dados lá do início. Após fazer essa atribuição, podemos criar outras 2 variáveis, a total geral, que mostra a soma de todos os idiomas, e a total de inglês, onde usamos o s.loc[‘en’] para pegar somente a contagem de inglês do nosso total por língua. Para saber qual o total do resto de línguas basta subtrair o total de inglês com o total geral, obtendo assim o total do resto:

total\_por\_lingua = tmdb['original\_language'].value\_counts()

total\_geral = sum(total\_por\_lingua)

total\_de\_ingles = total\_por\_lingua.loc['en']

total\_do\_resto = total\_geral - total\_de\_ingles

print(total\_de\_ingles, total\_do\_resto)



* + 1. A partir disso podemos criar um dict chamado dados que aloca duas chaves com 2 valores cada: língua, onde recebe o valor de ‘inglês’ e ‘outros’, e total, que recebe as variáveis de total de inglês e total do resto, convertendo esse dict em um df no final para trabalharmos com ele para fazer o gráfico:

dados = {

    'lingua': ['ingles', 'outros'],

    'total': [total\_de\_ingles, total\_do\_resto]

}

dados = pd.DataFrame(dados)

dados



* + 1. Decidimos fazer um gráfico de barras com sns para melhor visualização, passando como parâmetros sns.barplot(x=’língua’, y=’total’, data = dados), onde o x é a label, y são os valores e data é nosso df recém criado:

sns.barplot(x = 'lingua', y = 'total', data = dados)



* 1. Criamos uma variável que seleciona todas as línguas menos o inglês e criamos um sns.catplot() com ele para ver a importância relativa das outras línguas, uma vez que quando estava com o inglês incluso não conseguíamos ver praticamente nada além dele:

filmes\_sem\_lingua\_original\_em\_ingles = tmdb.query("original\_language != 'en'")

sns.catplot(x = 'original\_language', kind = 'count', data = filmes\_sem\_lingua\_original\_em\_ingles)



* 1. O que aprendemos:
     1. Comparar categorias;
     2. O que é uma Series;
     3. Instalar o seaborn;
     4. Transformar em dataframe com a função to\_frame();
     5. Remover o índice para gerar duas colunas usando a função reset\_index();
     6. Utilizar o Categorical do seaborn.

1. **Aula 5 – Continuando Com Visualização:**
   1. Notamos que na figura acima ficou tudo amontoado no eixo x e, diferentemente do sns.barplot() que nos permite personalizar como quisermos usando a plt por ser baixo nível, o catplot(), como é alto nível, o próprio sns que configura pra gente.
      1. Por conta disso, não conseguimos alterar o tamanho da figura como antes, precisando assim ir consultar a documentação do catplot e descobrir que existe um parâmetro chamado height e outro aspect, estando configurados como 5 e 1 respectivamente por default, ou seja, quadrado.
      2. Para alterar isso e expandir o eixo x do nosso gráfico, precisamos alterar o aspec. Colocando 2, ou seja, o dobro, já deixa nosso gráfico com a visualização bem melhor:

sns.catplot(x = 'original\_language', kind = 'count', data = filmes\_sem\_lingua\_original\_em\_ingles, aspect=2)



* + 1. Contudo notamos que ainda está fora de ordem, ao contrário do que estamos acostumados a ver, sendo ela do mais popular/visto/que apareceu para o que menos, uma vez que são variáveis categóricas nominais e não existe ordem de sequência.
    2. Entretanto, ao consultar a documentação, vimos que existe o parâmetro order, que define a ordem que nosso gráfico será exibido. Com isso podemos passar para ela o index da contagem de quantas vezes cada língua apareceu no nosso df e ele faz na ordem de frequência:

total\_por\_lingua\_de\_outros\_filmes = tmdb.query("original\_language != 'en'").original\_language.value\_counts()

sns.catplot(x = 'original\_language', kind = 'count', data = filmes\_sem\_lingua\_original\_em\_ingles, aspect=2, order=total\_por\_lingua\_de\_outros\_filmes.index)



* + 1. A cor ainda não está no ideal, então precisamos consultar a documentação de palette de cores do seaborn. Descobrimos que possuem palettes para cada tipo de gráfico, seja quantitativo, sequencial ou qual for.
    2. No nosso caso queremos sequencial, que ele vá do mais escuro para os mais frequentes até os mais claros para os menos.
    3. “Mako”, essa é a string da palette de cores que precisamos atribuir ao parâmetro palette=str do nosso gráfico:

sns.catplot(x = 'original\_language', kind = 'count', data = filmes\_sem\_lingua\_original\_em\_ingles, aspect=2, palette='mako', order=total\_por\_lingua\_de\_outros\_filmes.index)



* 1. Para saber mais: seaborn:
     1. Com o Seaborn, somos capazes de criar vários tipos de gráficos ou visualizações. Ao longo deste curso, aprendemos a fazer os gráficos de barras (histogramas) e diagrama de caixa (boxplot).
     2. Outro exemplo de gráfico que pode ser criado com essa biblioteca é o gráfico linear:

import seaborn as sns

sns.set(style="ticks")

# Load the example dataset for Anscombe's quartet

df = sns.load\_dataset("anscombe")

# Show the results of a linear regression within each dataset

sns.lmplot(x="x", y="y", col="dataset", hue="dataset", data=df,

col\_wrap=2, ci=None, palette="muted", height=4,

scatter\_kws={"s": 50, "alpha": 1})



* + 1. O dataframe utilizado para essa plotagem já está dentro da biblioteca do Seaborn!
    2. Outros tipos de gráfico com o seaborn: <http://seaborn.pydata.org/examples/>.
  1. O que aprendemos:
     1. Reescalar o gráfico;
     2. Ordenar o gráfico;
     3. Alterar os tons do gráfico.

1. **Aula 6 – Dados e Estatística:**
   1. Podemos juntar 2 arrays np com o np.append():

np.append(np.array([2.5] \* 10), np.array([3.5] \* 10))

* 1. Precisamos saber como utilizer e analisar os dados fazendo testes estatísticos, não podemos simplesmente aceitar usar a média ou mediana de um dado sem saber sua distribuição, como por exemplo 20 notas para 2 filmes diferentes, enquanto 1 tem 10 notas 2.5 e 10 notas 3.5 tendo média/ mediana 3, o 2 tem 10 notas 1 e 10 notas 5, também tendo média/mediana 3, ou seja, o mesmo resultado final para distribuições totalmente diferentes.
     1. Para melhorar isso utilizamos os gráficos, mas mesmo assim precisamos saber qual gráfico utilizar, uma vez que dependendo dele os dados podem ficar sobrepostos ou com uma visualização muito ruim. Por isso é importante conhecer várias bibliotecas e tipos de visualização de dados diferentes, dessa forma sempre sabendo qual a melhor forma de expressar um dado:

import numpy as np

filme1 = np.append(np.array([2.5] \* 10), np.array([3.5] \* 10))

filme2 = np.append(np.array([5] \* 10), np.array([1] \* 10))

print(np.mean(filme1), np.mean(filme2))

print(np.median(filme1), np.median(filme2))



sns.distplot(filme1)

sns.distplot(filme2)



plt.hist(filme1)

plt.hist(filme2)



sns.boxplot(filme1)

sns.boxplot(filme2)



plt.boxplot([filme1, filme2])



* + 1. O seaborn é muito bom para fazer gráficos quando ainda não temos os dados separados, servindo para a parte de exploração de dados.
    2. No caso do nosso df das notas, podemos pedir um sns.boxplot(x =’filmeId’, y = ‘nota’, data = notas), para que ele faça um boxplot de todas as notas para cada filme, mas tem o contra de ficar SUPER poluído:

sns.boxplot(x = 'filmeId', y = 'nota', data = notas)



* + - 1. Podemos filtrar esse gráfico com um query:

sns.boxplot(x = 'filmeId', y = 'nota', data = notas.query('filmeId == [1,2,3,4,5]'))



* + 1. Se quisermos saber o quanto os quartis desviam do padrão, utilizamos o: desvio padrão, ou std():

print(np.mean(filme1), np.mean(filme2))

print(np.std(filme1), np.std(filme2))

print(np.median(filme1), np.median(filme2))



* + 1. Agora temos 2 informações, a média de cada filme e a dispersão das notas, ou seja, o quanto elas variam.
  1. Para saber mais: Desvio padrão:
     1. Em estatística, dispersão de dados pode ser chamada de variabilidade ou espalhamento e são dados estatísticos para mostrar uma determinada variação de dados, e com esses dados é possível obter a variância, o desvio padrão e a amplitude interquartil.
     2. Mas vamos demonstrar um exemplo de desvio padrão em uma sala de aula pensando na quantidade de aulas assistidas e de alunos que vão a aula. Teremos o seguinte bloco:



* + 1. Temos o desvio padrão quando temos 2 alunos com a quantidade 4 de aulas assistidas! E nas outras quantidades de aulas assistidas temos apenas 1 aluno participando, sendo assim temos um desvio quando temos 2 alunos com a quantidade de 4 aulas assistidas, porque ele quebra o padrão de ter um aluno por aula.
    2. Referência: <https://pt.wikipedia.org/wiki/Dispers%C3%A3o_estat%C3%ADstica>.
  1. O que aprendemos:
     1. Comparar a média de filmes;
     2. Juntar arrays com o np.append();
     3. O que é a dispersão de dados;
     4. Desvio padrão.

**Curso II - Introdução A Análise De Series Temporais:**

Link para o drive:

-<https://drive.google.com/drive/folders/1aSBybSUq8ecD0YfYPIyiuK0r-Qw0OeHc>

Link para a documentação das libs que trabalharemos:

- <http://pandas.pydata.org/pandas-docs/stable/>

- <https://matplotlib.org/>

- <http://seaborn.pydata.org/introduction.html>

1. **Aula 1 – Tendência das Vendas:**
   1. Podemos ver a quantidade de dados nulos em um df usando o df.isna().sum(), assim ele mostra a somatória de dados nulos em cada coluna. Para saber apenas o número total, sem separar por colunas podemos colocar mais um sum() no final:

alucar = pd.read\_csv('alucar.csv')

print(f'Quantidade de dados nulos: {alucar.isna().sum().sum()}')



* + 1. Como vamos trabalhar com series temporais é interessante ver quais são os tipos dos nossos dados com o df.dtypes e converter as datas df.data = pd.to\_datetime(df.data), dessa forma ficando bem melhor para trabalharmos:

alucar.dtypes



alucar.mes = pd.to\_datetime(alucar.mes)

alucar.dtypes



* + 1. Para criar um gráfico com essas informações, podemos utilizar o sns.lineplot(x = x, y = y, data = data), com ele será plotado um gráfico de linha mostrando, seguindo nosso df, como foram as vendas de acordo com as datas:

sns.lineplot(x = 'mes', y = 'vendas', data = alucar)



* 1. Aperfeiçoando o gráfico:
     1. Para melhorar a visualização gráfica, podemos utilizar diversos style guide disponibilizados sendo um deles o urban institute visualization style guide: <https://urbaninstitute.github.io/graphics-styleguide/>.
     2. Nesses guias podemos ver qual deve ser os tamanhos de títulos, labels, set de paletas de cores, dentre outras boas práticas relacionadas a gráficos.
     3. Para começar nosso aperfeiçoamento podemos colocar um grid no background do gráfico com sns.set\_style(‘darkgrid’) e mudar a paleta de cores com sns.set\_palette(‘Accent’):

sns.set\_palette('Accent')

sns.set\_style('darkgrid')

sns.lineplot(x = 'mes', y = 'vendas', data = alucar)



* + 1. Apesar de ter ficado melhor, sempre que criamos o gráfico é exibido o local de armazenamento da memória no topo, como um título, para remover basta atribuir a uma variável. Além disso, podemos ainda aumentar o tamanho do nosso gráfico com o ax.figure.set\_size\_inches(12, 6), deixando a visualização bem mais clara:

ax = sns.lineplot(x = 'mes', y = 'vendas', data = alucar)

ax.figure.set\_size\_inches(12, 6)



* + 1. Podemos ainda atribuir o título e labels, alterando também o tamanho da fonte e, no caso do título, a localização, que vem por padrão ‘center’ e podemos mudar para ‘left’ e ‘right’:

ax.set\_title('Vendas Alucar de 2017 e 2018', loc = 'left', fontsize = 18)

ax.set\_xlabel('Meses', fontsize = 14)

ax.set\_ylabel('Vendas (R$)', fontsize = 14)



* + - 1. Apesar do endereço da memória ter desaparecido, começou a aparecer outra coisa, que para remover, basta atribuir ax = ax:



* + 1. Isso é uma time series, uma coleção de observações que geralmente são feitas através de um espaço de tempo, representando cada ponto.
  1. Para saber mais - time series:
     1. Aprendemos que time series é uma coleção de observações ou dados, geralmente feita através de um tempo sequencial, que podendo ser segundos, minutos, dias, meses, anos, entre outros.
     2. Tipicamente, esse tempo é espaçado entre cada ponto de dados, que podemos medir o crescimento das vendas, como visto nesta primeira aula.
     3. Link para o blog da alura falando mais sobre: <https://blog.alura.com.br/crescimento-proporcional-entre-linhas-time-series-utilizando-pandas/>.
  2. O que aprendemos:
     1. Carregamos um arquivo csv com as vendas da Alucar de 2017 e 2018;
     2. Examinamos o arquivo através de funções do pandas, para descobrir a quantidade de linhas e colunas presentes com o comando alucar.shape;
     3. Verificamos se havia dados nulos com o comando alucar.isna().sum();
     4. Alteramos o tipo do mês de object para datetime com o comando alucar['mes'] = pd.to\_datetime(alucar['mes']);
     5. Importamos as bibliotecas necessárias para gerar um gráfico das vendas, porém ao plotar, o gráfico e as labels não estavam num tamanho adequado e sem um título;
     6. Aperfeiçoamos o gráfico incluindo uma nova palette de cor, incluindo título e labels com tamanho adequado descrevendo melhor do que se trata nosso gráfico.

1. **Aula 2 – Decomposição e Autocorrelação:**
   1. Decompondo As Vendas:
      1. A partir do nosso gráfico anterior podemos ver que a cada mês que passa as vendas aumentam, mas podemos extrair ainda mais informação desses dados decompondo essas vendas, ou seja, subtrair as vendas desse mês das do mês passado, a do mês passado com a do anterior e assim por diante. Para fazer isso existe a função df.s.diff(), onde ele faz automaticamente esse processo. Adicionamos uma nova coluna com essas informações:

alucar['aumento'] = alucar.vendas.diff()

alucar.head()



* + 1. Fizemos um gráfico para expressar isso usando as mesmas configurações que o outro, mudando apenas o título e a ylabel:

sns.set\_palette('Accent')

sns.set\_style('darkgrid')

ax = sns.lineplot(x = 'mes', y = 'aumento', data = alucar)

ax.figure.set\_size\_inches(12, 6)

ax.set\_title('Aumento das Vendas Alucar de 2017 e 2018', loc = 'left', fontsize = 18)

ax.set\_xlabel('Meses', fontsize = 14)

ax.set\_ylabel('Aumento', fontsize = 14)

ax = ax



* + - 1. Para não ficar copiando e colando todo esse código sempre que a gente for plotar um gráfico colocamos ele em uma função chamada plotar(), assim passamos só as informações e ele plota pra gente:

def plotar(titulo: str, labelx: str, labely: str, x: str, y: str, dataset):

  sns.set\_palette('Accent')

  sns.set\_style('darkgrid')

  ax = sns.lineplot(x = x, y = y, data = dataset)

  ax.figure.set\_size\_inches(12, 6)

  ax.set\_title(titulo, loc = 'left', fontsize = 18)

  ax.set\_xlabel(labelx, fontsize = 14)

  ax.set\_ylabel(labely, fontsize = 14)

  ax = ax

plotar('Aumento das Vendas Alucar de 2017 e 2018', 'Meses', 'Aumento', 'mes', 'aumento', alucar,)



* + 1. Com ambos os gráficos, tanto da quantidade de vendas, quanto do aumento, estão com bons resultados, mostrando que realmente estão subindo. Podemos extrair ainda mais informação vendo qual é a aceleração que esse crescimento está tendo. Esse processo é basicamente o mesmo que fizemos para o aumento, porém faremos utilizando o aumento, ou seja: alucar.aumento.diff():

alucar['aceleracao'] = alucar.aumento.diff()

alucar.head()



* + 1. Ao plotar esse gráfico vemos que nossa escala vai somente de 1 a 3, ou seja, não tivemos, em momento algum, valores negativos nas nossas vendas:

plotar('Aceleração das Vendas da Alucar de 2017 e 2018', 'Meses', 'Aceleração', 'mes', 'aceleracao', alucar)



* 1. Observação e tendência:
     1. Apesar de ter ficado bem claro e conseguirmos ver os gráficos legalzinho, seria ainda mais interessante se conseguíssemos colocar todos eles em um único plot, uma única imagem com 3 plots separados de cada tipo de análise. Para isso utilizamos o matplotlib.pyplot e criamos uma figura bem grande, criando uma área para subplots atribuída a uma variável e passando as configurações para ela:

plt.figure(figsize = (16, 12))      # Seta o tamanho da imagem

ax = plt.subplot(3, 1, 1)     # Atribui o subplot a ax e seta a loc do primeiro gráfico

ax.set\_title('Análise de Vendas da Alucar de 2017 e 2018',

             fontsize = 18, loc = 'left')   # Seta título, fonte e localização

sns.lineplot(x = 'mes', y = 'vendas', data = alucar)  # Primeiro Gráfico

plt.subplot(3, 1, 2)      # Localização do segundo gráfico, divide em 3 partes, diz que ocupa a parte toda, diz qual é a parte em que plotará

sns.lineplot(x = 'mes', y = 'aumento', data = alucar) # Segundo Gráfico

plt.subplot(3, 1, 3)                # Localização do terceiro gráfico

sns.lineplot(x = 'mes', y = 'aceleracao', data = alucar)  # Terceiro gráfico

ax = ax



* + 1. Como vamos fazer outros gráficos em subplots dessa maneira, criamos uma função para isso como para plotar somente 1:

def plot\_comparacao(x: str, y1: str, y2: str, y3: str, dataset, titulo: str):

  plt.figure(figsize = (16, 12))

  ax = plt.subplot(3, 1, 1)

  ax.set\_title(titulo, fontsize = 18, loc = 'left')

  sns.lineplot(x = x, y = y1, data = dataset)

  plt.subplot(3, 1, 2)

  sns.lineplot(x = x, y = y2, data = dataset)

  plt.subplot(3, 1, 3)

  sns.lineplot(x = x, y = y3, data = dataset)

  ax = ax

plot\_comparacao('mes', 'vendas', 'aumento', 'aceleracao', alucar, 'Análise de Vendas da Alucar de 2017 e 2018')

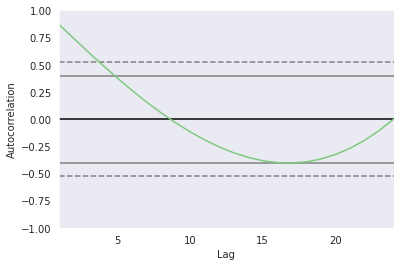


* 1. Autocorrelação:
     1. Se quiser descobrir que os aumentos dessa semana estão ligados com os da semana passada, podemos fazer a autocorrelação, que faz a medida de correlação entre as observações de uma serie temporal.
     2. A correlação possuí níveis, onde o máximo é 1 e o mínimo é -1, ou seja, se a correlação for muito alta, o valor que encontraremos dessa função é 1, o contrário também é verdade. Ainda temos valores intermediários que variam de 0.99 até -0.99.
     3. A biblioteca que faz isso pra gente é a autocorrelation\_plot, da pandas.plotting:

from pandas.plotting import autocorrelation\_plot

* + 1. Para ver a autocorrelação, chamamos ela e colocamos a series que queremos autocorrelacionar, gerando automaticamente um gráfico dela:

autocorrelation\_plot(alucar.vendas)



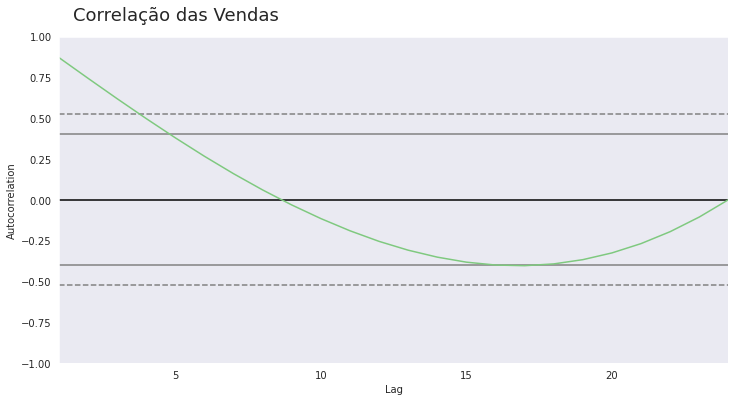
* + - 1. Podemos ver que existe uma correlação significativa no lag 1 que vai decaindo com o tempo até terminar na correlação nula.
    1. Podemos melhorar esse gráfico com algumas configurações que, por ser um gráfico advindo do pandas, que utiliza o plt para plotar, precisamos passar os parâmetros que o plt utiliza para fazer a configuração:

ax = plt.figure(figsize=(12,6))

ax.suptitle('Correlação das Vendas', fontsize = 18, x = 0.26, y = 0.95)

autocorrelation\_plot(alucar.vendas)

ax = ax



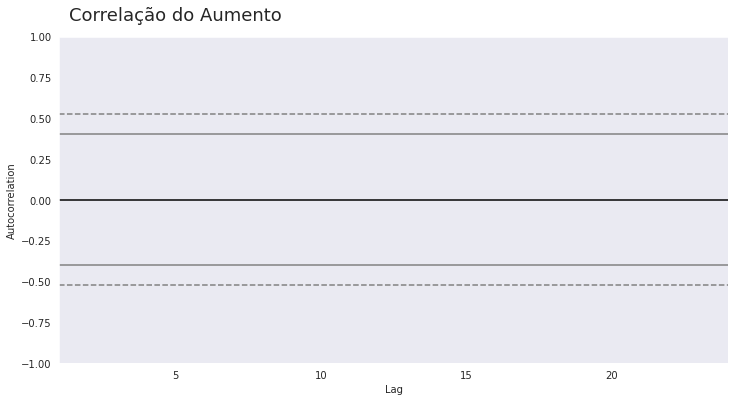
* + 1. Fizemos o mesmo para o aumento, mas recebemos um gráfico vazio, isso porque nosso primeiro índice, ou seja, o 0, recebe um valor nulo, portanto precisamos passar o índice 1: para que vejamos o gráfico:

ax = plt.figure(figsize=(12,6))

ax.suptitle('Correlação do Aumento', fontsize = 18, x = 0.26, y = 0.95)

autocorrelation\_plot(alucar.aumento)

ax = ax

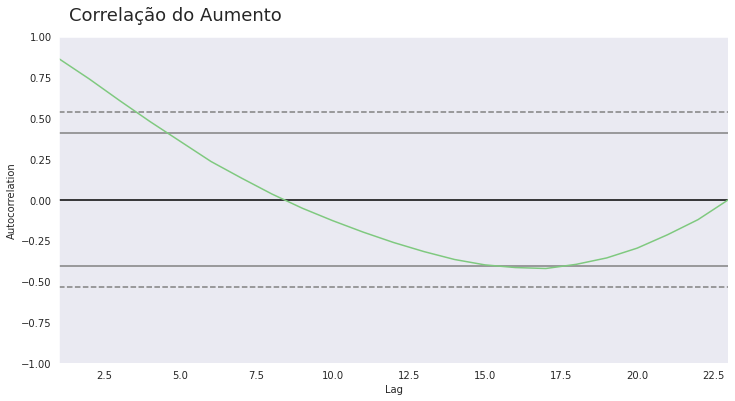


ax = plt.figure(figsize=(12,6))

ax.suptitle('Correlação do Aumento', fontsize = 18, x = 0.26, y = 0.95)

autocorrelation\_plot(alucar.aumento[1:])

ax = ax



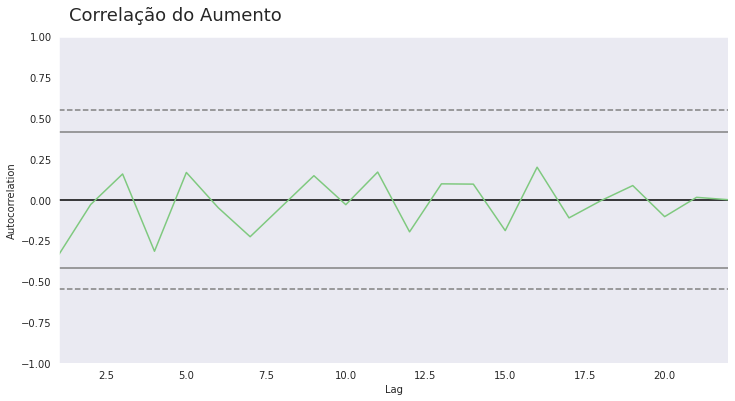
* + 1. O mesmo vale para a aceleração, que inicia no índice 2:

ax = plt.figure(figsize=(12,6))

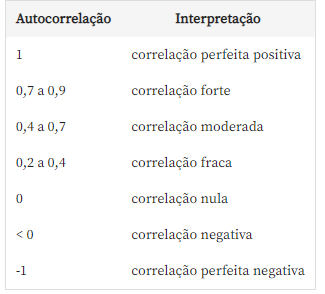
ax.suptitle('Correlação do Aumento', fontsize = 18, x = 0.26, y = 0.95)

autocorrelation\_plot(alucar.aceleracao[2:])

ax = ax

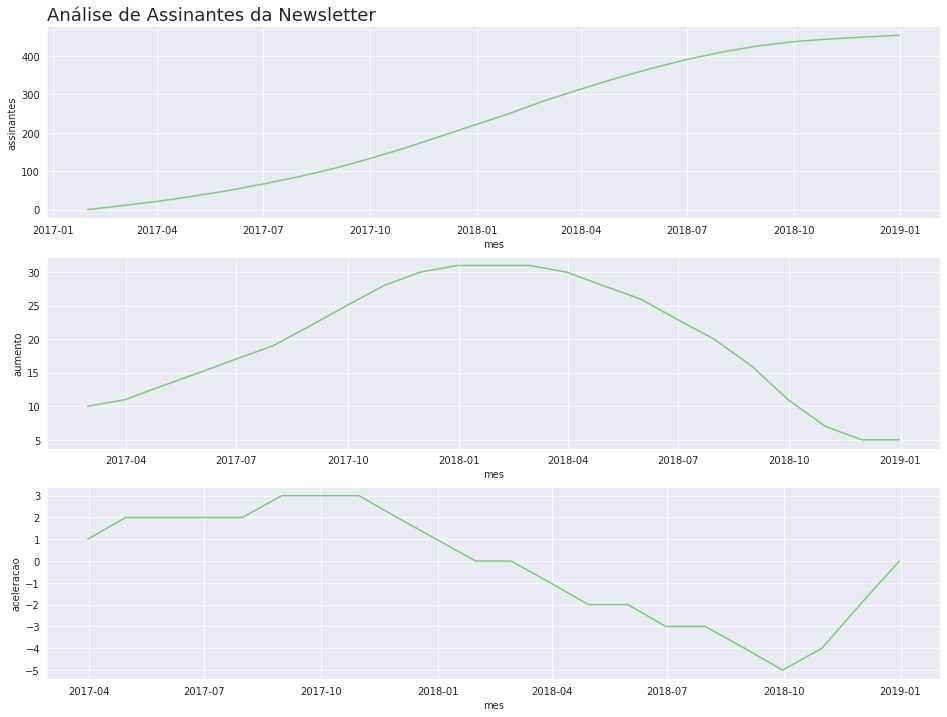


* + 1. A partir desses gráficos podemos concluir que os 7 primeiros dias tem uma correlação muito grande com a semana passada e com o passar do tempo decresce.
  1. Para saber mais – Autocorrelação:
     1. A função de autocorrelação é a principal ferramenta de diagnóstico para análise de time series;
     2. Autocorrelação significa a correlação de valores de uma mesma variável ordenados no tempo.
     3. Para melhor interpretação do gráfico, podemos analisar a tabela abaixo:

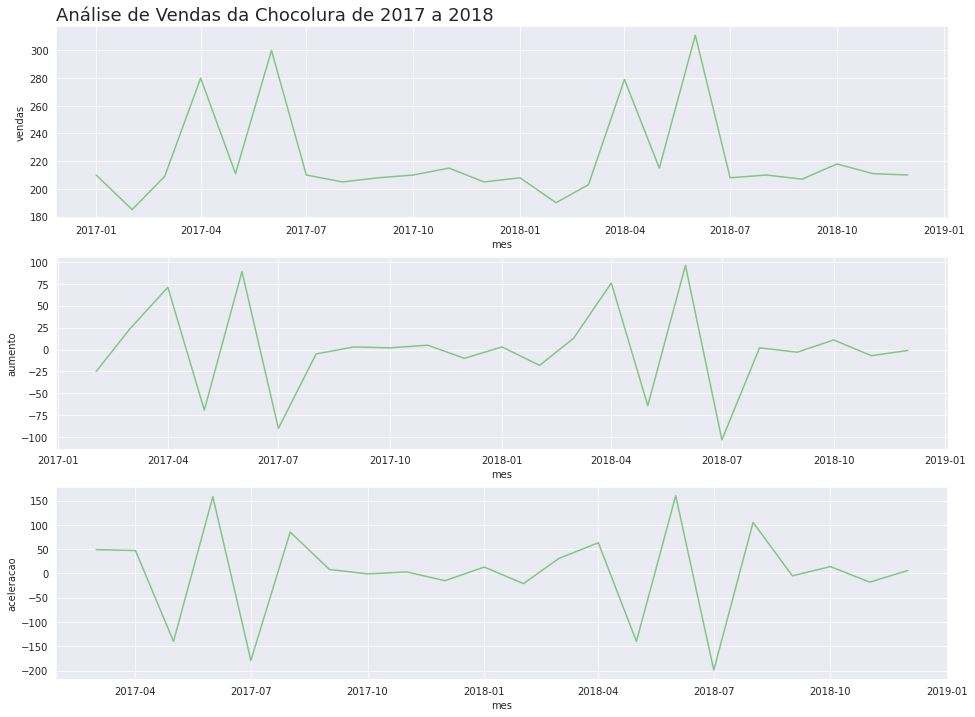


* 1. O que aprendemos:
     1. Aplicamos a técnica de Decomposição de uma time series, para mensurar o crescimento mês a mês;
     2. Utilizamos a função diff() para decomposição das vendas para criar o aumento, e do aumento para descobrir a aceleração;
     3. Executamos a função de Autocorrelação para descobrir o nível de correlação das vendas, do aumento e da aceleração;
     4. Criamos uma função chamada plotar() e plot\_comparacao() para geração de gráficos padronizados, evitando código duplicado.

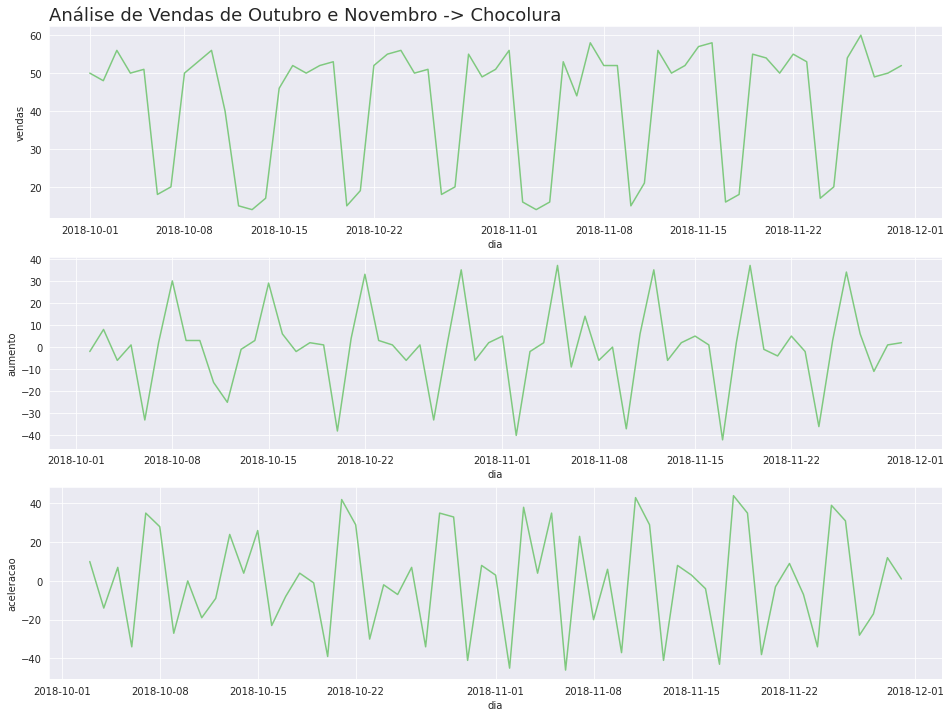
1. **Aula 3 – Importância Da Decomposição e Sazonalidade:**
   1. Analisando Assinantes:
      1. Pegamos os assinantes e fizemos a mesma coisa que com dos valores da alucar:



* + 1. Com esses gráficos podemos ver que teve uma tendência de crescimento dos assinantes, mas que o aumento chegou ao seu pico em determinado ponto e depois começou a cair muito, já a aceleração, podemos notar que houve um crescimento negativo muito grande, chegando a -5.
  1. Sazonalidade e Chocolate:
     1. Fizemos as mesmas análises para a venda de chocolate da chocoalura:



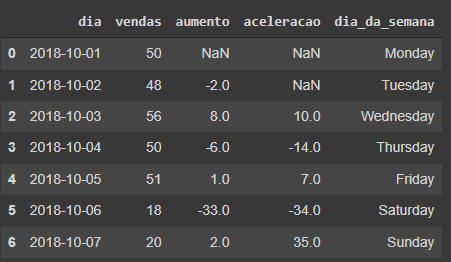
* + 1. Podemos observar que se trata de uma sazonalidade, ou seja, um padrão repetitivo dentro de um período de tempo, ou seja, sempre no mês 4 e sempre no mês 6 são as datas que a empresa vende mais chocolate.
  1. Investigando a Sazonalidade:
     1. Fizemos as mesmas análises para as vendas diárias da chocolura de outubro e novembro:



* + 1. Diferente daquele outro padrão sazonal ocorrendo em 2 pontos do ano, esse podemos ver um padrão sazonal muito repetitivo.
    2. Para investigar o que aconteceu com esses dados, criamos uma nova variável e chamamos ela de dia da semana. Utilizamos a função dt.day\_name() a partir da nossa variável dia, onde ela nomeia os dias corretamente a partir da data da variável dia:

vendas\_por\_dia['dia\_da\_semana'] = vendas\_por\_dia.dia.dt.day\_name()

vendas\_por\_dia.head(7)



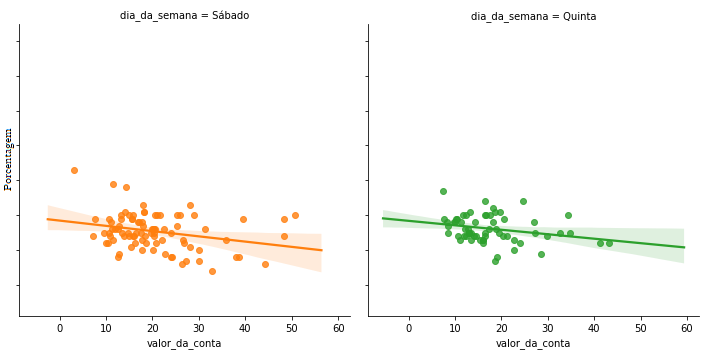
* + 1. Traduzimos os dias, e pegamos a média de cada variável agrupada por dia da semana para ver o que pode ter acontecido:

vendas\_agrupadas = vendas\_por\_dia.groupby(vendas\_por\_dia.dia\_da\_semana).mean().round()

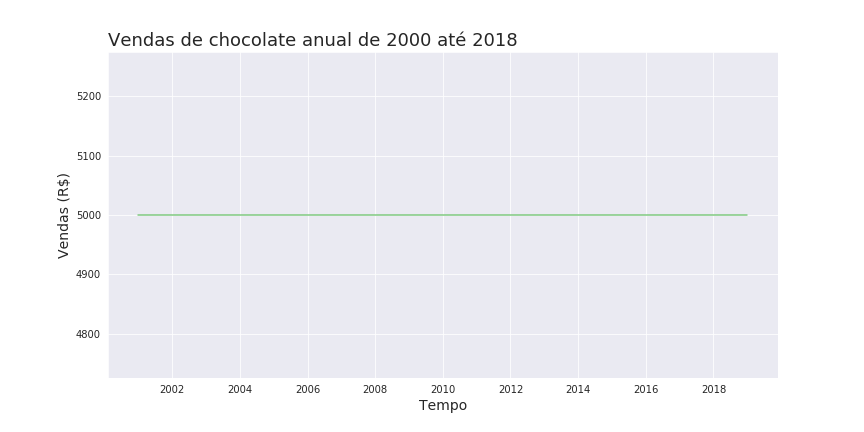
vendas\_agrupadas



* + 1. Através dessa tabela das médias podemos notar que os dias que tem menos vendas e que inclusive a aceleração e aumento médios são negativos é no sábado, sendo o domingo também bem ruim comparado ao resto da semana, mesmo sem ter valores negativos. Isso pode nos dizer que talvez por alguma regra de negócios a loja pode abrir por menos tempo, ou por algum motivo ter menos vendas no sábado e domingo do que no resto da semana, causando esse efeito de sazonalidade.
  1. Para saber mais - Quando não aplicar análise de time series:
     1. Sabemos que time series ou series temporais são uma forma de organizar no tempo as informações quantitativas e que podemos extrair muitas informações deste tipo de dado. Porém, nem sempre podemos aplicar a análise de time series.
     2. Abaixo temos duas recomendações de quando não aplicar analise de time series:



* + 1. Analisando os gráficos acima, não temos em nenhum dos eixos, a variável tempo.
    2. A principal variável em uma time series é o tempo. Sendo assim, a ausência desta variável impossibilita análise de séries temporais.
    3. As variáveis ou valores são constantes:



* + 1. Analisando o gráfico acima de uma loja que vende 5000 caixas de chocolates há 18 anos, não podemos aplicar análises de séries temporais para extrair mais informações dos dados.
  1. O que aprendemos:
     1. Vimos a importância da técnica de Decomposição na análise dos assinantes da newsletter da Alucar;
     2. Analisamos as vendas da Chocolura e descobrimos um padrão repetitivo no movimento das vendas dentro de um período de tempo fixo, na qual é chamado de Sazonalidade;
     3. Examinamos as vendas de uma determinada loja da Chocolura nos meses de Outubro e Novembro de 2018, e descobrimos que também havia uma sazonalidade;
     4. Investigamos o que causava a sazonalidade desta loja.

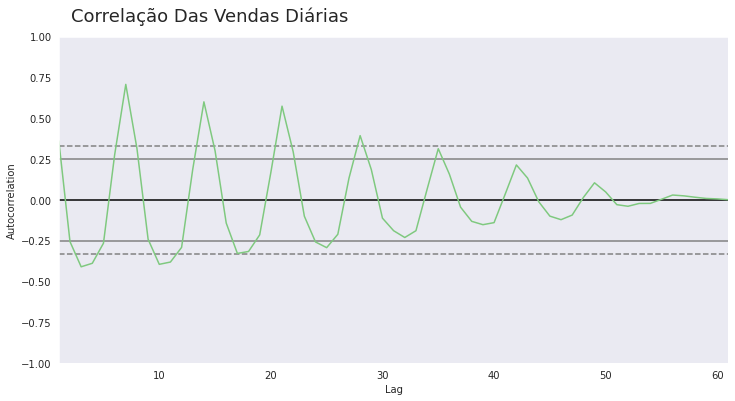
1. **Aula 4 – Normalização e Statsmodels:**
   1. Autocorrelação Das Vendas:
      1. Fizemos a correlação das vendas dos dados de vendas diárias e notamos que existe uma correlação muito forte no início, mas que vai caindo e se aproxima de 0 com o tempo:

ax = plt.figure(figsize = (12,6))

ax.suptitle('Correlação Das Vendas Diárias', fontsize = 18, x = 0.3, y = 0.95)

autocorrelation\_plot(vendas\_por\_dia.vendas)

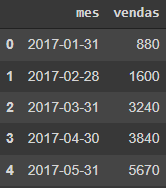
ax = ax



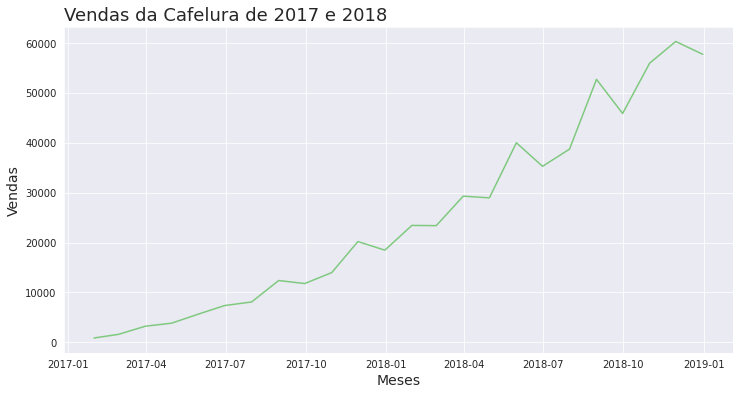
* 1. Normalização:
     1. Fizemos a normalização dos dados da cafelura dividindo as vendas pela quantidade de dias de final de semanas:

cafe = pd.read\_csv('cafelura.csv')

cafe.head()

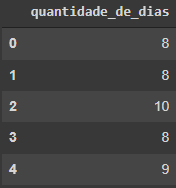


plotar('Vendas da Cafelura de 2017 e 2018', 'Meses', 'Vendas', 'mes', 'vendas', cafe)



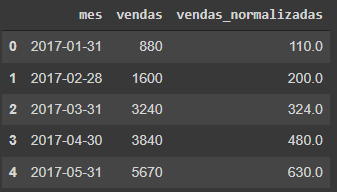
quantidade\_de\_dias\_de\_fds = pd.read\_csv('dias\_final\_de\_semana.csv')

quantidade\_de\_dias\_de\_fds.head()



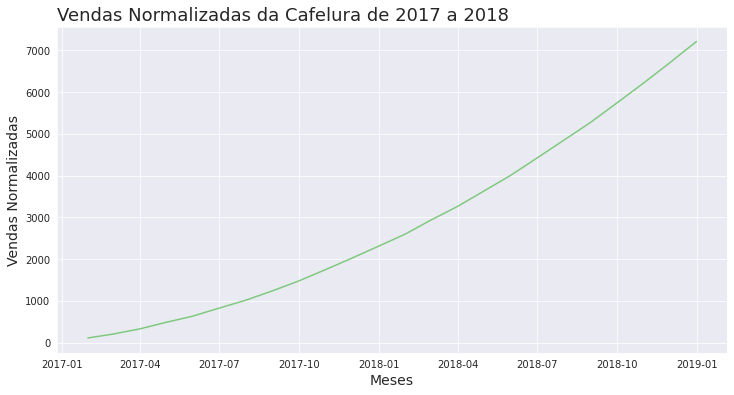
cafe['vendas\_normalizadas'] = cafe.vendas / quantidade\_de\_dias\_de\_fds.quantidade\_de\_dias.values

cafe.head()



plotar('Vendas Normalizadas da Cafelura de 2017 a 2018', 'Meses',

       'Vendas Normalizadas', 'mes', 'vendas\_normalizadas', cafe)



plt.figure(figsize = (12, 8))

ax = plt.subplot(2, 1, 1)

ax.set\_title('Vendas da Cafelura de 2017 e 2018')

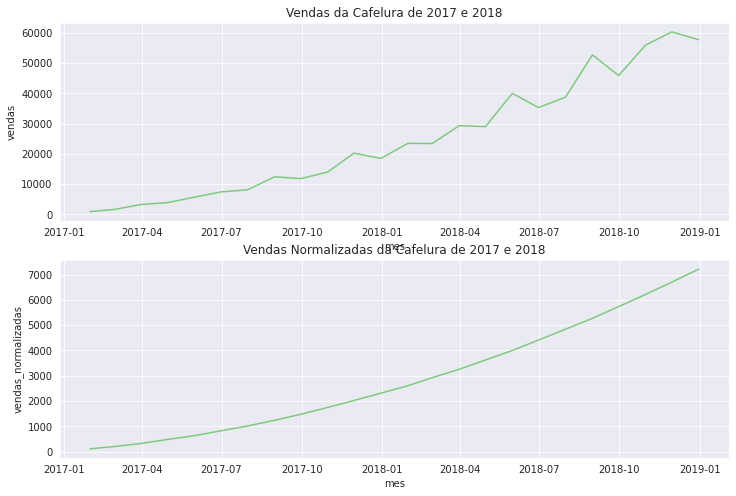
sns.lineplot(x = 'mes', y = 'vendas', data = cafe)

ax = plt.subplot(2, 1, 2)

ax.set\_title('Vendas Normalizadas da Cafelura de 2017 e 2018')

sns.lineplot(x = 'mes', y = 'vendas\_normalizadas', data = cafe)

ax = ax

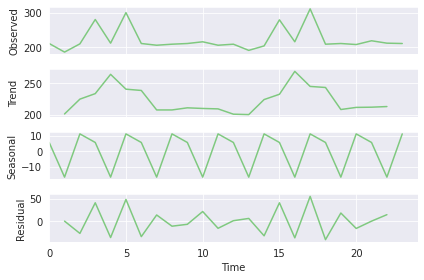


* 1. Statsmodel:
     1. É um exemplo de como podemos usar uma função para visualizar tudo aquilo que queremos. Podemos fazer todos os gráficos: observação, tendencia, sazonalidade e ruído, de uma única vez com o seasonal\_decompose da lib statsmodels.tsa.seasonal, onde, ao atribuir o resultado em uma variável podemos criar um gráfico com ela:

from statsmodels.tsa.seasonal import seasonal\_decompose

resultado = seasonal\_decompose([choco.vendas], freq = 3)

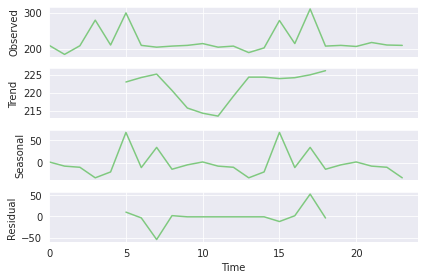
ax = resultado.plot()



* + 1. A frequência é o que determina o quanto vamos ver de cada elemento, mas quanto maior for, menos ruído e tendência conseguimos ver:

resultado = seasonal\_decompose([choco.vendas], freq = 10)

ax = resultado.plot()



* + 1. Podemos ainda separar cada um desses elementos em uma variável e criar um gráfico nosso, utilizando nossas funções feitas bem acima:

observacao = resultado.observed

tendencia = resultado.trend

sazonalidade = resultado.seasonal

ruido = resultado.resid

data = ({

    'observacao': observacao,

    'tendencia': tendencia,

    'sazonalidade': sazonalidade,

    'ruido': ruido

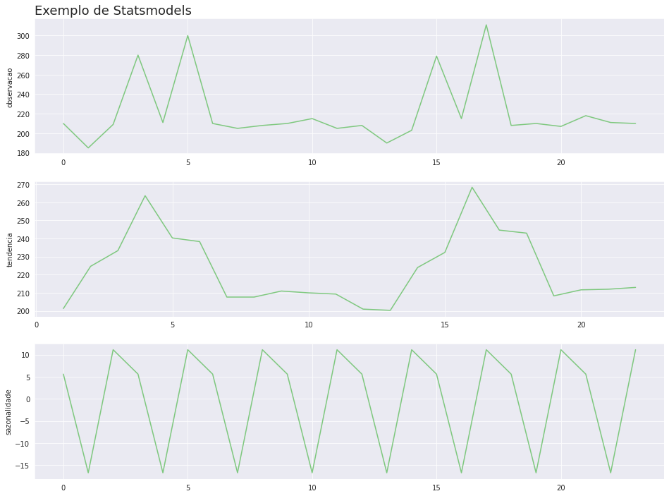
})

resultado = pd.DataFrame(data)

resultado.head()



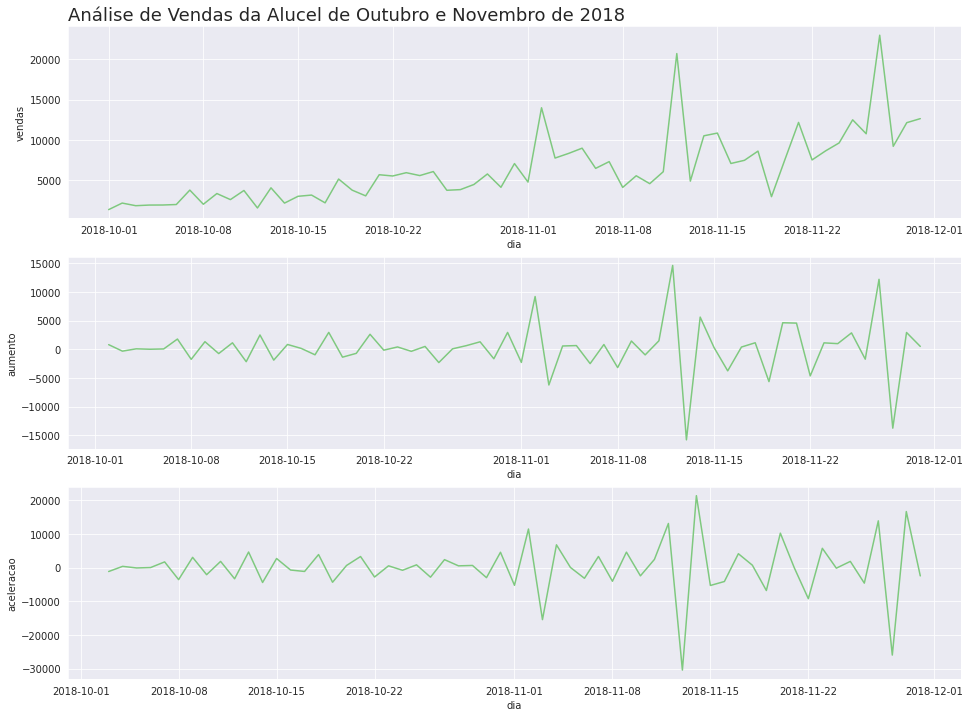
plot\_comparacao(resultado.index, 'observacao', 'tendencia', 'sazonalidade', resultado, 'Exemplo de Statsmodels')



* 1. Para saber mais - Encontrei dados nulos... e agora?:
     1. Se tentarmos plotar um gráfico que possui dados nulos, uma mensagem de erro será exibida informando que não temos tipos numéricos para agregar.
     2. Pensando nisso, dentre as inúmeras possíveis soluções de contornar esse problema, criei [este caderno com 5 possíveis soluções para contornar a falta dos dados](https://colab.research.google.com/drive/1UZFN5Jke6gFgYLNMbsU7eBJ9Pk9TdpIm).
     3. Dentre as soluções listadas no notebook acima, temos:
        1. Preenchendo os dados nulos com o valor médio da coluna;
        2. Preenchendo os dados nulos com um valor específico;
        3. Deletando os dados nulos do DataFrame;
        4. Preenchendo um dado nulo com o valor anterior ou posterior;
        5. Interpolando dados nulos com a média;
        6. [Neste link, você pode acessar a documentação do pandas e encontrar outras soluções para dados nulos ou perdidos](https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html).
  2. O que aprendemos:
     1. Aplicamos a função de Autocorrelação nas vendas, no aumento e na aceleração das vendas diárias e vimos que havia uma correlação entre elas;
     2. Executamos uma técnica de normalização de time series para minimizar as frequências pela quantidade de dias de finais de semana de cada mês;
     3. Analisamos uma time series importando da biblioteca statsmodels.tsa.seasonal a função seasonal\_decompose, que nos mostra o que é a nossa observação, tendência, sazonalidade e ruído de uma só vez.

1. **Aula 5 – Ruídos e Média Móvel:**
   1. Ruído De Uma Time Series:
      1. Fizemos a plotagem dos gráficos das variáveis como anteriormente, mas com os dados do alucel agora. Note que temos muito ruído e não conseguimos dizer com precisão se a empresa está crescendo ou não:

plot\_comparacao('dia', 'vendas', 'aumento', 'aceleracao', alucel, 'Análise de Vendas da Alucel de Outubro e Novembro de 2018')



* + 1. Por isso precisamos normalizar os dados, removendo o ruído do nosso gráfico.
  1. Média Móvel:
     1. Apesar de ter muito ruído, podemos tirar usando a técnica de média móvel, que tem como função minimizar o ruído de uma série temporal através de médias móveis, ou seja, ela pega dois pontos ou mais (como por exemplo fazendo a média de 7 em 7 dias), faz a média e vai marcando os pontos para minimizar essa dispersão da frequência que está muito alta, ou seja, o ruído.
     2. Para executar essa técnica usamos a função df.s.rolling(n).mean() do pandas, assim ele faz a média usando a variável escolhida em ‘s’ em uma frequência de ‘n’ pontos:

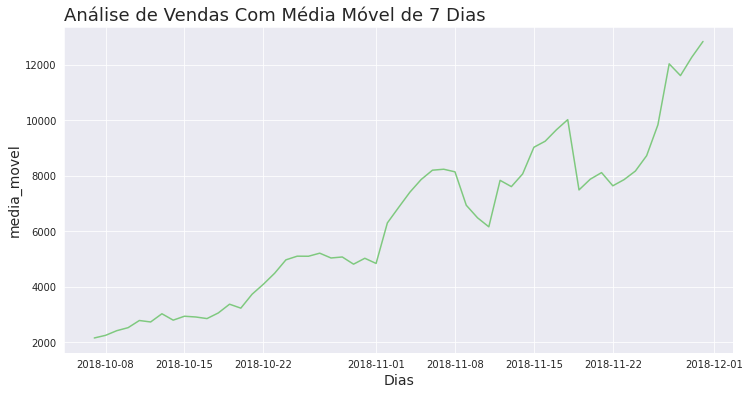
alucel['media\_movel'] = alucel.vendas.rolling(7).mean()

alucel.head(7)



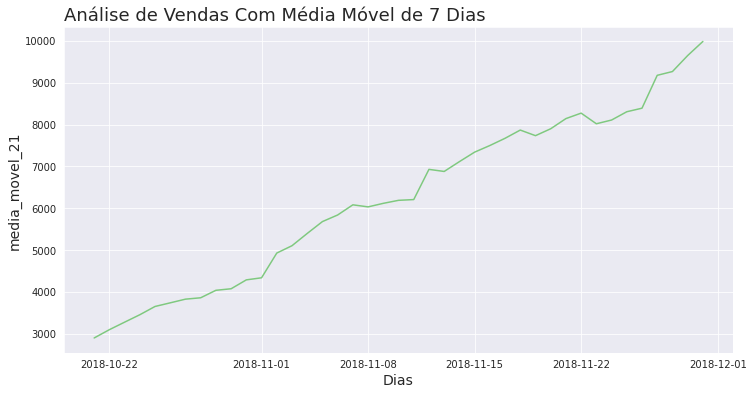
* + 1. Como podemos ver na tabela, no sétimo dia tem a média de vendas de todas as 7 observações, ficando assim no gráfico:

plotar('Análise de Vendas Com Média Móvel de 7 Dias', 'Dias', 'media\_movel','dia', 'media\_movel', alucel)



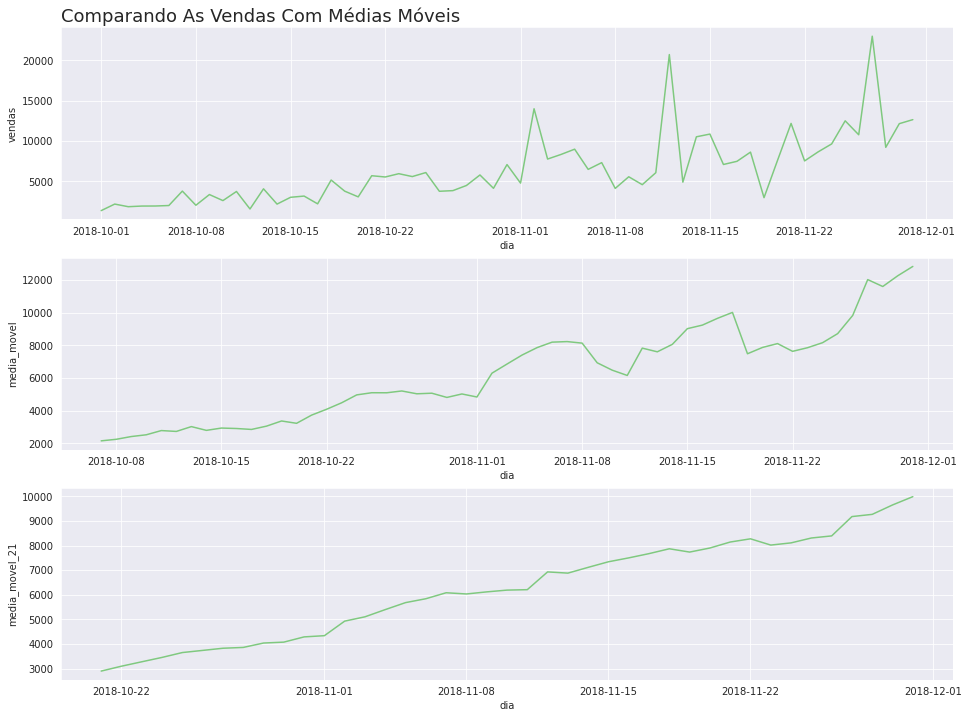
* + 1. Não iremos fazer sempre de 7 dias, tudo vai depender da quantidade de amostras e do tipo de amostras que estamos trabalhando. Fizemos agora com 21 dias, olha como normalizou ainda mais:

plotar('Análise de Vendas Com Média Móvel de 7 Dias', 'Dias', 'media\_movel\_21','dia', 'media\_movel\_21', alucel)



* + 1. Porém, apesar de normalizar mais, quanto maior o rolling, mais informações perdemos, ou seja, parece que a empresa está crescendo apesar de algumas quedas, mas se olharmos o primeiro gráfico novamente veremos tiveram picos bem altos tanto pra cima quanto pra baixo. Portanto, o ideal é sempre normalizar com média móvel, mas sem perder a essência dos dados, portanto com média móvel de 7 dias está ótimo, nesse caso. Os 3 plots:

plot\_comparacao('dia', 'vendas', 'media\_movel', 'media\_movel\_21', alucel,'Comparando As Vendas Com Médias Móveis')



* 1. O que aprendemos:
     1. Aprendemos que um componente presente na maioria das Time Series é o ruído;
     2. Vimos que podemos minimizar os ruídos de uma time series aplicando a técnica da média móvel;
     3. Criamos um gráfico com a média móvel de 7 e 21 dias e comparamos com nossa observação.