**Estatística Com Python**

**Parte I – Frequências e Medidas:**

Todos os arquivos serão baixados, mas deixo o link do drive de qualquer forma: <https://drive.google.com/drive/folders/1MoAf3dLinpNBXVGIIyetaY5qfSwZ31Gt>.

1. **Aula 1 – Qual Seu Tipo de Dado:**
   1. Dependendo do tipo de dados análises diferentes serão feitas. Importante saber quais são os tipos.
      1. Quantitativos: Expressa quantidade de dados. Contagem, mensurações. Ex.:
         1. Idade;
         2. Renda;
         3. Altura.
      2. Qualitativos: Expressa qualidade dos dados. Ex.:
         1. UF;
         2. Sexo;
         3. Cor;
         4. Anos de Estudo.
      3. Podemos dividir os qualitativos em ordinais e nominais, e os quantitativos em discretos e contínuos.
   2. O que aprendemos:
      1. A identificar a versão de uma biblioteca do Python;
      2. A ler um dataset no formato CSV e criar um DataFrame pandas, com o conteúdo deste dataset;
      3. A identificar e classificar as variáveis de um dataset como quantitativas ou qualitativas;
      4. Que as variáveis qualitativas se dividem em ordinais e nominais;
      5. Que as variáveis quantitativas se dividem em discretas e contínuas.
2. **Aula 2 – Distribuição de Frequência:**
   1. S.value\_counts(): conta a quantidade de vezes que uma variável aparece na series. Podemos passar o parâmetro normalize = True, onde ele devolve a porcentagem de cada variável, onde ele soma elas e divide cada um pela soma:

dados['Sexo'].value\_counts(normalize=True) \* 100



* + 1. Podemos usar a porcentagem e a frequência para criar um df de distribuição e frequência qualitativas:

dist\_freq\_qualitativas = pd.DataFrame({'Frequência': frequencia, 'Porcentagem (%)': percentual})



* + 1. Df.rename\_axis(‘nome\_index’, axis=’columns’/’rows’, inplace=True): Renomeia a coluna do index:

dist\_freq\_qualitativas.rename(index= {0: 'Masculino', 1: 'Feminino'}, inplace=True)

dist\_freq\_qualitativas.rename\_axis('Sexo', axis='columns', inplace=True)

dist\_freq\_qualitativas



* 1. Pd.crosstab(variável\_linha, variável\_coluna): Cria um df onde faz o cruzamento dos dados da linha e da coluna passadas. No caso utilizamos cor e sexo, então ele irá mostrar na tabela quantas pessoas do sexo masculino tiveram de cada cor, bem como os femininos:

sexo = {0: 'Masculino',

        1: 'Feminino'}

cor = {0: 'Indígena',

        2: 'Branca',

        4: 'Preta',

        6: 'Amarela',

        8: 'Parda',

        9: 'Sem declaração'}

frequencia = pd.crosstab(dados.Sexo, dados.Cor)

frequencia.rename(index=sexo, columns=cor, inplace=True)

frequencia



* + 1. Podemos passar o mesmo parâmetro (normalize=True) que para value\_counts() e já receber o valor das porcentagens:

frequencia = pd.crosstab(dados.Sexo, dados.Cor, normalize=True) \* 100

frequencia



* + 1. Podemos ainda fazer algo bem mais legal. Podemos passar o valor de outra variável do nosso dataset para ser analisado nesses cruzamentos, ou seja, se escolhermos média da renda, por exemplo, ao invés de mostrar quantos homens e mulheres existem de cada cor, ele passa a exibir, em média, quanto os homens e mulheres ganham de renda. Para isso utilizaremos os parâmetros aggfunc=análise\_estat’ e values = dados[‘index’]:

frequencia = pd.crosstab(dados.Sexo, dados.Cor, aggfunc = 'mean', values = dados.Renda)

frequencia



* 1. Pd.cut(variável, bins=[’distribuição’], labels=labels, include\_lowest=True/False): criamos uma lista com o intervalo colocado nos bins a partir da variável escolhida fazendo relação com as labels que determinamos, como no caso da renda, onde temos a classificação de A a E da renda populacional dos nossos dados analisados. Utilizamos essa função para mostrar quais pessoas estão em qual classificação a partir da renda:

classes = [0, 1576, 3152, 7880, 15760, 200000]

labels = ['E', 'D', 'C', 'B', 'A']

pd.cut(dados.Renda, bins = classes, labels = labels, include\_lowest=True)



* + 1. Porém Podemos notar que ele classifica indivíduo por indivíduo. Para saber a frequência podemos colocar o cut dentro do pd.value\_counts(), sabendo quantas pessoas estão em cada classificação:

frequencia = pd.value\_counts(pd.cut(dados.Renda,

                                    bins = classes,

                                    labels = labels,

                                    include\_lowest=True))

frequencia



* + 1. Para saber o percentual, basta colocar o normalize=True e multiplicar por 100, como estávamos fazendo com os outros:

percentual = pd.value\_counts(pd.cut(dados.Renda,

                                    bins = classes,

                                    labels = labels,

                                    include\_lowest=True), normalize=True) \* 100

percentual



* + 1. Criamos a tabela de frequência e porcentagem com pd.DataFrame():

dist\_freq\_quantitativas\_personalizadas = pd.DataFrame({'Frequência': frequencia,

                                                       'Porcentagem (%)': percentual})

dist\_freq\_quantitativas\_personalizadas



* + 1. E alteramos a ordem do index para crescente, ou seja, de A para baixo com o sorted\_index(ascending=True):

dist\_freq\_quantitativas\_personalizadas.sort\_index(ascending=False)



* 1. Biblioteca numpy possuí diversas fórmulas matemáticas que podemos usar.
     1. Com isso podemos fazer vários cálculos como log10 dentre vários outros, permitindo com que façamos cálculos como tamanho de classes:



* + - 1. N = valor de linhas do nosso df. Basta atribuir o df.shape[0] em n.
    1. Tendo a fórmula nós podemos executar a conta:

n = dados.shape[0]

k = 1 + (10/3) \* np.log10(n)

k = int(k.round(0))

* + - 1. Arredondamos porque não tem como fazer 17.26.... classes, apenas números inteiros.
    1. Com isso podemos fazer um cut passando k como as bins para ter essa classe de amplitude fixa. De resto, a criação da frequência é a mesma:

frequencia = pd.value\_counts(pd.cut(x = dados.Renda, bins = k, include\_lowest = True), sort = False)

frequencia



* + 1. O mesmo vale para o percentual:

percentual = pd.value\_counts(pd.cut(x = dados.Renda, bins = k, include\_lowest = True), sort = False, normalize=True) \* 100

percentual



dist\_freq\_quantitativas\_amplitude\_fixa = pd.DataFrame({'Frequência': frequencia, 'Porcentagem (%)': percentual})

dist\_freq\_quantitativas\_amplitude\_fixa



* 1. Sns.distplot(dados, kde=bool): cria um histograma com a variável do df passada.
     1. O kde é a exibição de densidade do gráfico. Quando true mostra a linha traçada em cima das colunas no fundo, quando false, só as colunas.
     2. Podemos atribuir esse gráfico a uma variável (ax, por exemplo) para ficar com fácil acesso.
     3. A partir disso podemos configurar o tamanho de exibição do gráfico com ax.set\_size\_inches(x, y).
     4. O Título com ax.set\_titles(‘titulo’, fontsize=n).
     5. E as labels com ax.set\_xlabels(‘label’, fontsize=n) e ax.set\_ylabels(‘label’, fontsize=n).
     6. Exemplo:

ax = sns.displot(dados.Altura, kde=True)

ax.figure.set\_size\_inches(12,6)

ax.set\_titles('Distribuição de Frequências - Altura - KDE', fontsize = 18)

ax.set\_xlabels("Metros", fontsize = 14)

ax



Kde = false:



* + 1. Podemos obter um resultado parecido com o pandas usando a função dados.Altura.hist.(bins=n\_barras, figsize=(x,y)):

dados.Altura.hist(bins = 50, figsize=(12,6))



* + 1. Com pandas ainda podemos fazer um df[‘variavel’].plot.bar(width=n, color=’color’, alpha=n). Alpah é a transparência da cor das barras:

dist\_freq\_quantitativas\_personalizadas.Frequência.plot.bar(width = 1, color = 'blue', alpha = 0.2)



* 1. O que aprendemos:
     1. A criar distribuições de frequências (tabelas de frequências) com a função value\_counts() do pandas;
     2. A criar distribuições de frequências, com o cruzamento de duas variáveis, utilizando a função crosstab() do pandas;
     3. A criar distribuições de frequências, com classes personalizadas, utilizando as funções value\_counts() e cut() conjuntamente;
     4. A utilizar a regra de Sturges para obter um número de classes ótimo para determinado tamanho de amostra;
     5. A plotar o histograma, que é a representação gráfica de uma distribuição de frequências.

1. **Aula 3 – Medidas de Tendência Central:**
   1. A média é o ponto de equilíbrio da variável.
      1. As vezes não é o melhor dado para representar um grupo, uma vez que, no caso da renda, se tiver muita gente ganhando pouco e pouca gente ganhando muito, a média não representa essa variável, pois apesar de ser uma média grande, o desvio padrão e a distribuição dos dados são altos, tendo uma baixa representatividade. Cálculo da média:



* + 1. A média nada mais é do que a soma dos valores dividido pelo seu n total, ou seja:

(8 + 10 + 4 + 8 + 6 + 10 + 8) / 7



* + 1. Felizmente não precisamos fazer isso na mão desse jeito sempre, uma vez que seria impossível fazer isso com 70k de dados, por exemplo. Principalmente se forem floats.
    2. Nesse caso o pandas nos ajuda e muito com suas função sum() e seu shape. Com eles podemos pegar apenas uma coluna de informação e passar para o sum que irá somar todos os valores, em seguida podemos fazer a divisão dessa soma pela quantidade de observações do nosso dado, ou seja, quantidade de linhas dele. Para isso usamos o atributo shape[0], que nos retorna exatamente essa informação. No final obtemos o mesmo resultado que feito na mão, assim:

df.Fulano.sum() / df.shape[0]



* + 1. Ou então podemos fazer de uma forma ainda MUITO mais fácil: simplesmente pedir a média dessa coluna para o pandas com o mean():

df.Fulano.mean()



* + 1. Utilizando esse conhecimento podemos utilizar nossos dados e pegar a média de todas as rendas, ou então usar a função de agrupamento do pandas e fazer a média diretamente a partir de uma variável para esse grupo df.groupby(‘variável\_para\_agrupar\_por’)[‘variável\_analisar’].mean():

dados.groupby(['Sexo'])['Renda'].mean()



* 1. Mediana é o valor que divide a series exatamente no meio, por exemplo, a idade de 10 pessoas, organizamos em ordem crescente e vemos qual é a idade que divide exatamente no meio, com os mais novos pra um lado e os mais velhos para o outro.



* + 1. Cálculos – Identificando Elemento:
       1. Quando n for ímpar:



* + - 1. Quando n for par:



* + 1. Cálculos – Obtendo a Mediana:
       1. Quando n for ímpar:



* + - 1. Quando n for par:



* + 1. Exemplo no python:

df = pd.DataFrame(data = {'Fulano': [8, 10, 4, 8, 6, 10, 8],

                          'Beltrano': [10, 2, 0.5, 1, 3, 9.5, 10],

                          'Sicrano': [7.5, 8, 7, 8, 8, 8.5, 7]},

                  index = ['Matemática', 'Português', 'Inglês',

                           'Geografia', 'História', 'Física', 'Química'])

df.rename\_axis('Matérias', axis = 'columns', inplace = True)

df



notas\_fulano = df.Fulano

notas\_fulano = notas\_fulano.sort\_values()

notas\_fulano = notas\_fulano.reset\_index()

n = notas\_fulano.shape[0]

elemento\_md = int((n + 1) / 2)

notas\_fulano.loc[elemento\_md - 1]



Ou então:

notas\_fulano.median()



* + 1. Com n par:

notas\_beltrano = df.Beltrano.sample(6, random\_state=101)

notas\_beltrano



notas\_beltrano.median()



* 1. A moda é sempre o que mais aparece, ou seja, no caso das notas do df acima, temos que a nota modal de fulano é 8, uma vez que ela aparece 3 vezes.
     1. Se tivermos 2 modas chamamos de bimodal.
     2. Mais de 3, multimodal.
     3. Para pegar as modas use df.mode(). Será retornado um df com as modas:

df.mode()



* + 1. Multimodal:

exemplo = pd.Series([1, 2, 2, 3, 4, 4, 5, 6, 6])

exemplo.mode()



* 1. Relação entre média, mediana e moda:
     1. Moda maior que mediana maior que média: Assimetria à direita:

ax = sns.distplot(dados.query('Renda < 20000').Renda)

ax.figure.set\_size\_inches(12,6)

ax



* + - 1. Tirando a prova nos dados:

moda = dados.Renda.mode()[0]

moda



mediana = dados.Renda.median()

mediana



media = dados.Renda.mean()

media



moda < mediana < media



* + 1. Todas iguais (com uma margem de erro): Simétrica:

ax = sns.distplot(dados.Altura)

ax.figure.set\_size\_inches(12,6)

ax



* + - 1. Prova nos dados:

moda = dados.Altura.mode()

moda



mediana = dados.Altura.median()

mediana



media = dados.Altura.mean()

media



* + 1. Média menor que mediana menor que moda: Assimetria à esquerda:

ax = sns.distplot(dados['Anos de Estudo'], bins = 17)

ax.figure.set\_size\_inches(12,6)

ax



moda = dados['Anos de Estudo'].mode()[0]

moda



mediana = dados['Anos de Estudo'].median()

mediana



media = dados['Anos de Estudo'].mean()

media



moda > mediana > media



* 1. O que aprendemos:
     1. A calcular as principais medidas de tendência central: média aritmética, mediana e moda;
     2. A identificar características importantes de uma distribuição, como a presença de assimetria e sua direção a partir da relação entre as medidas de tendência central.

1. **Aula 4 – Medidas Separatrizes:**
   1. Há uma série de medidas de posição semelhantes na sua concepção à mediana, embora não sejam medidas de tendência central. Como se sabe, a mediana divide a distribuição em duas partes iguais quanto ao número de elementos de cada parte. Já os quartis permitem dividir a distribuição em quatro partes iguais quanto ao número de elementos de cada uma; os decis em dez partes e os centis em cem partes iguais.
      1. S.quantile(.25, .75): Devolve os quartis todos, dependendo do valor passado no parâmetro. Se não passar nada, ele devolve o 0.5, ou seja, a mediana.
      2. Podemos ainda passar uma lista para receber todos os quartis de uma vez:

dados.Renda.quantile([0.25, 0.5, 0.75])



* + 1. Utilizamos a mesma função para fazer os decis, mas ao invés dos parâmetros que passamos antes, fazemos uma list comprehensions com um for para repetir esse processo 9x:

dados.Renda.quantile([i / 10 for i in range(1, 10)])



* + - 1. Desse modo temos que 10% das pessoas estão abaixo de 350 enquanto 90 estão acima. O mesmo raciocínio serve para o resto dos decis.
    1. A mesma lógica se aplica aos percentis, mas ao invés de 10 colocamos 100 em tudo:

dados.Renda.quantile([i / 100 for i in range(1, 100)])



* + - 1. A principal diferença é na quantidade de informações que temos, já que ele varia de 1% em 1%. Podemos inclusive notar que 4% das pessoas estão abaixo de 50, mas ao 5% estão abaixo de 100 e assim por diante.
    1. Podemos criar um histograma e visualizar em gráfico o que vimos em números, passando para os parâmetros sns.distplot(dados, hist.\_kws= {‘cumulative’:True}, kde\_kws = {‘cumulative’:True}, bins=10). Ao passar o bins ele vai mostrar 10 barras representando os decis:

ax = sns.distplot(dados.Idade, hist\_kws= {'cumulative':True},

                  kde\_kws = {'cumulative':True}, bins = 10)

ax.figure.set\_size\_inches(12,6)

ax.set\_title('Distribuição de Frequências Acumuladas', fontsize=18)

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

ax.set\_xlabel('Anos', fontsize=14)

ax



* 1. Box-plot são representações gráficas bem importantes pois mostra muito da distribuição que estamos estudando, como informações de posição, dispersão, simetria, dados discrepantes, dentre outros, e são construídos a partir de medidas separatrizes.



* + 1. Podemos criar sns.boxplot(x, y~~opt~~, data=data, orient=’h’) passando mais de 1 parâmetro, como x e y, por exemplo, sendo 2 dados diferentes. Parâmetro orient=’h’ para deixar os gráficos horizontais:

ax = sns.boxplot( x = 'Altura', y = 'Sexo', data = dados, orient = 'h')

ax.figure.set\_size\_inches(12, 4)

ax.set\_title('Altura', fontsize=18)

ax.set\_xlabel('Metros', fontsize=14)

ax



* + 1. Conseguimos passar uma query para fazer uma seleção de dados, como no caso da renda onde não conseguimos ver nada por causa do outlier:

ax = sns.boxplot( x = 'Renda', data = dados, orient = 'h')

ax.figure.set\_size\_inches(12, 4)

ax.set\_title('Altura', fontsize=18)

ax.set\_xlabel('Metros', fontsize=14)

ax



ax = sns.boxplot( x = 'Renda', data = dados.query('Renda < 10000'), orient = 'h')

ax.figure.set\_size\_inches(12, 4)

ax.set\_title('Altura', fontsize=18)

ax.set\_xlabel('Metros', fontsize=14)

ax



* + 1. Podemos ainda colocar 2 dados como acima:

ax = sns.boxplot( x = 'Renda', y = 'Sexo', data = dados.query('Renda < 10000'), orient = 'h')

ax.figure.set\_size\_inches(12, 4)

ax.set\_title('Altura', fontsize=18)

ax.set\_xlabel('Metros', fontsize=14)

ax



* + 1. Comparação de simetria:

