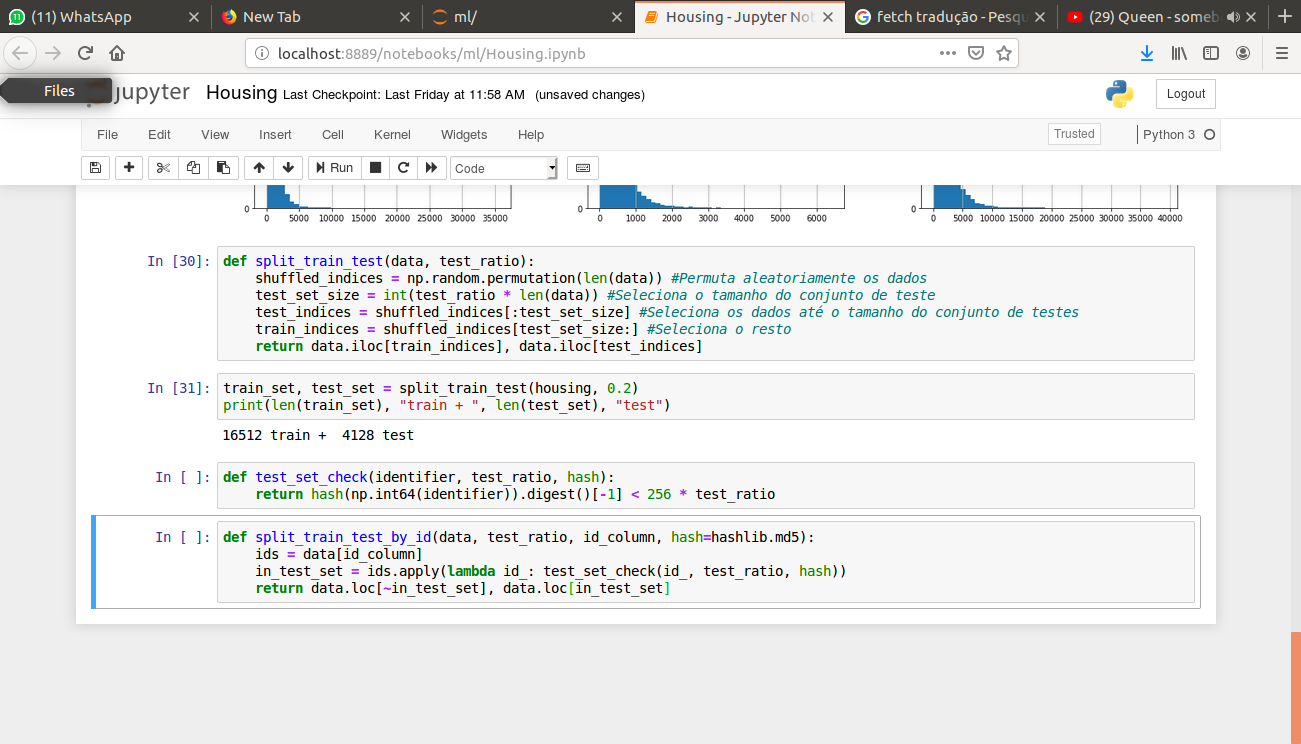
**Machine Learning**

**Criando um conjunto de teste**

Isto funciona, porém não é perfeito. Se rodarmos o programa novamente, ele irá gerar um diferente conjunto de teste. Ao longo do tempo, você, ou seu algoritmo de ML, vai poder observar todo o dataset, que é aquilo que você quer evitar.

Uma solução é salvar o conjunto de teste na primeira execução e depois carregá-lo em execuções subsequentes. Uma outra opção é setar a seed do gerador de números aleatórios (e.g., np.random.seed(42)) antes de chamar np.random.permutation(), assim, sempre irá gerar as mesmas permutações.

Mas ambas dessas soluções irão quebrar a próxima vez que você trazer um dataset atualizado. Uma solução comum é usar o identificador de casa instância para decidir se deve ou não ir pro test set (assumindo que cada instância tenha um um identificador imutável e único). Por exemplo, você pode computar uma parte de cada identificador, deixando apenas o último byte, e colocar esta instância no test set se este valor for menor ou igual a 51 (~20% de 256). Isto garante que o test set se manterá constante após multiplas execuções, mesmo que você atualize-o. O novo test set irá conter 20% das novas instâncias, mas não irá conter nenhuma instância que fora previamente utilizada no training set.

Infelizmente, o dataset de housing não apresenta uma coluna de identificação. A solução mais simples é usar o índice da linha como ID

*housing\_with\_id = housing.reset\_index() # Adiciona uma coluna de índice*

*train\_set, test\_set = split\_train\_test\_by\_id(housing\_with\_id, 0.2, "index")*

Se você utilizar o índice da linha como um identificador único, você necessita fazer com que os novos dados sejam adicionados no final do dataset, e que nenhuma linha seja deletada. Se isto não for possível, então você pode tentar usar as features mais estáveis para criar um identificador único. Por exemplo, a latitude e a longitude dos distritos são guarantidos de serem estáveis por milhões de anos, então você pode combiná-los com o ID.

*housing\_with\_id = housing.reset\_index() # Adiciona uma coluna de índice*

*housing\_with\_id["id"] = housing["longitude"]\*1000 + housing["latitude"]*

*train\_set, test\_set = split\_train\_test\_by\_id(housing\_with\_id, 0.2, "index")*

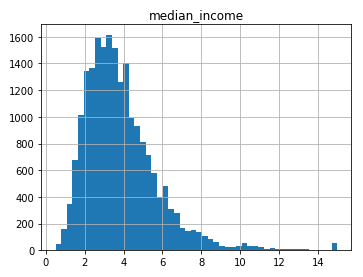
Scikit-Learn provê algumas funções para seprar datasets em múltiplos subconjuntos de várias formas. A função mais simples é train\_test\_split, a qual faz basicamente a mesma coisa que a função split\_train\_test definiu anteriormente, com alguns recursos adicionais. Primeiramente, existe um parâmetro random\_state que permite que você defina o gerador de seed aleatório como explicado previamente, e segundo, você pode passar múltiplos datasets com um número identico de linhas, e irá dividí-los com os mesmos índices (isto é muito útil, por exemplo, se você tem DataFrame separado para labels):

from sklearn.model\_selection import train\_test\_split

train\_set, test\_set = train\_test\_split(housing, test\_size=0.2, random\_state=42)

Até agora nós apenas consideramos métodos aleatórios de amostragem. Isso geralmente é tranquilo se o seu dataset é grande o suficiente (especialmente relativo ao número de atributos), mas se ele não é, você corre o risco de introduzir bias significantes. Por exemplo, quando uma empresa de pesquisas decide chamar 1000 pessoas para perguntá-las algumas questões, eles não simplesmente pegam 1000 pessoas aleatórias, eles tentam assegurar que essas 1000 pessoas são uma parcela representativa de toda a população. Por exemplo, a população america é composta de 51,3% de mulheres e 48,7% de homens, então uma pesquisa bem conduzida nos Estados UNidos tenta manter essa razão na amostragem: 513 mulheres e 487 homens. Isto é chamado *Stratified Sampling*: a população é dividida em subgrupos homogêneos chamados de stata, e o número exato de instâncias é amostrado para cada stratum de formar a garantir a representativa da população geral. Se eles usassem amostragens randômicas, teria uma chance de 12% de criar um test set com menos de 49% de mulheres ou mais de 54%. De qualquer mdo, os resultados apresentariam muitas bias.

Vamos supor que você conversou com experts que o disseram que a median income é um atributo muito importante para predizer median housing prices. Você pode querer assegurar que o test set é um representativo de várias categorias de incomes em todo o dataset. Já que o median income é um atributo numeral contínuo, você primeiramente deve criar um atributo de categoria de income. Vamos dar mais uma olhada no histograma de median income:



A maioria dos valores de renda média se concentram entre $20000 e $50000, mas algumas rendas medias vão bem mais além de $60000. É importante que haja um número suficiente de instâncias em seu dataset para cada stratum, senão a esimativa da importância do stratum pode ser prejudicada. Isso significa que você não deve ter muitos strata’s, mas cada stratum deve ser grande o suficiente. O próximo código cria um atributo de categoria de renda dividindo a renda média por 1.5 (para limitar o número de categorias de renda), arredondá-lo usando /ceil/ (para haver categorias discretas) e depois fundir todas as categorias maiores que 5 na categoria 5:

housing["income\_cat"] = np.ceil(housing["median\_income"] / 1.5)

housing["income\_cat"] = where(housing["income\_cat"] < 5, 5.0, inplace=True)

Agora estamos prontos para criar uma amostragem stratificada baseada na categoria de renda. Para isso, você pode usar a classe do Scikit-Learn StratifiedShuffleSplit.