Aprendizado de Máquina

Projeto de Aprendizado de Máquina Ponta a Ponta

Prof. Regis Pires Magalhães

regismagalhaes@ufc.br - http://bit.ly/ufcregis

GÉRON, Aurélien; **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn, Keras & TensorFlow:** Conceitos, Ferramentas e Técnicas para a Construção de Sistemas Inteligentes. 2ª Ed. Alta Books, 2021.

PARTE I - Os conceitos básicos do aprendizado de máquina

1. O Cenário do Aprendizado de Máquina
2. **Projeto de Aprendizado de Máquina Ponta a Ponta**
3. Classificação
4. Treinando Modelos
5. Máquinas de Vetores de Suporte
6. Árvores de Decisão
7. Aprendizado Ensemble e Florestas Aleatórias (Bagging, Random Forests, Boosting, Stacking)
8. Redução de Dimensionalidade (PCA, Kernel PCA, LLE)
9. Técnicas de Aprendizado Não Supervisionado (Clusterização, Misturas de gaussianas)

PARTE II - Redes Neurais e Aprendizado Profundo

10. Introdução às Redes Neurais Artificiais com a Biblioteca Keras
11. Treinando Redes Neurais Profundas
12. Modelos Customizados e Treinamento com a Biblioteca TensorFlow
13. Carregando e Pré-processando Dados com a TensorFlow
14. Visão Computacional Detalhada das Redes Neurais Convolucionais
15. Processamento de Sequências Usando RNNs e CNNs
16. Processamento de Linguagem Natural com RNNs e Mecanismos de Atenção
17. Aprendizado de Representação e Aprendizado Gerativo com Autoencoders e GANs
18. Aprendizado por Reforço
19. Treinamento e Implementação de Modelos TensorFlow em Larga Escala

Passos

1. Analisar o panorama geral.
2. Obter os dados.
3. Identificar e visualizar os dados para obter informações úteis.
4. Preparar os dados para os algoritmos do aprendizado de máquina.
5. Selecionar e treinar um modelo.
6. Aperfeiçoar o modelo.
7. Apresentar a solução.
8. Disponibilizar em produção, monitorar e fazer a manutenção do sistema.

Trabalhando com Dados Reais

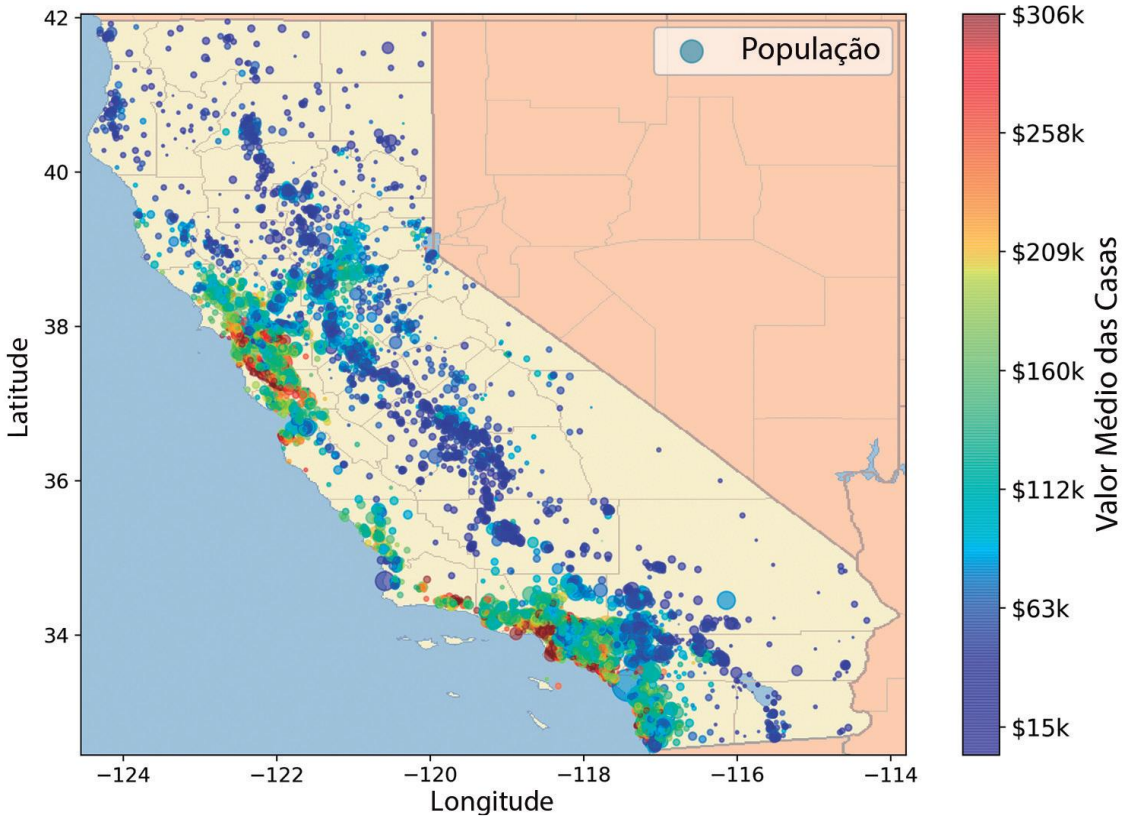
- Repositórios populares de open data
  - UC Irvine Machine Learning Repository
    - http://archive.ics.uci.edu/ml/
  - Conjunto de dados no Kaggle
    - https://www.kaggle.com/datasets
  - Conjunto de Dados no AWS da Amazon
    - https://registry.opendata.aws/
- Metaportais de dados (repositórios open data)
  - Data Portals
    - http://dataportals.org/
  - OpenDataMonitor
    - http://opendatamonitor.eu/
  - Quandl
    - http://quandl.com/

Trabalhando com Dados Reais

- Outras páginas que listam muitos repositórios populares de open data
  - Lista de conjuntos de dados de aprendizado de máquina do Wikipedia
    - https://homl.info/
  - Quora.com
    - https://homl.info/10
  - Conjuntos de dados em subseção do Reddit
    - https://www.reddit.com/r/datasets

Trabalhando com Dados Reais

- Estaremos usando aqui:
  - Conjunto de dados dos preços de imóveis da Califórnia, armazenado no repositório StatLib.
    - Baseado no censo da Califórnia de 1990.



Analisando o Panorama Geral

- Tarefa:
  - criar um modelo de preços para setor imobiliário usando os dados do censo do estado da Califórnia.
- Dados com indicadores como:
  - população, renda média e preço médio do imóvel para cada grupo de bairros (regiões).
    - um grupo de bairros geralmente comporta uma população de 600 a 3 mil pessoas.
- Objetivo:
  - realizar uma previsão do preço médio do imóvel em qualquer região, considerando todos os outros indicadores.

Analisando o Panorama Geral

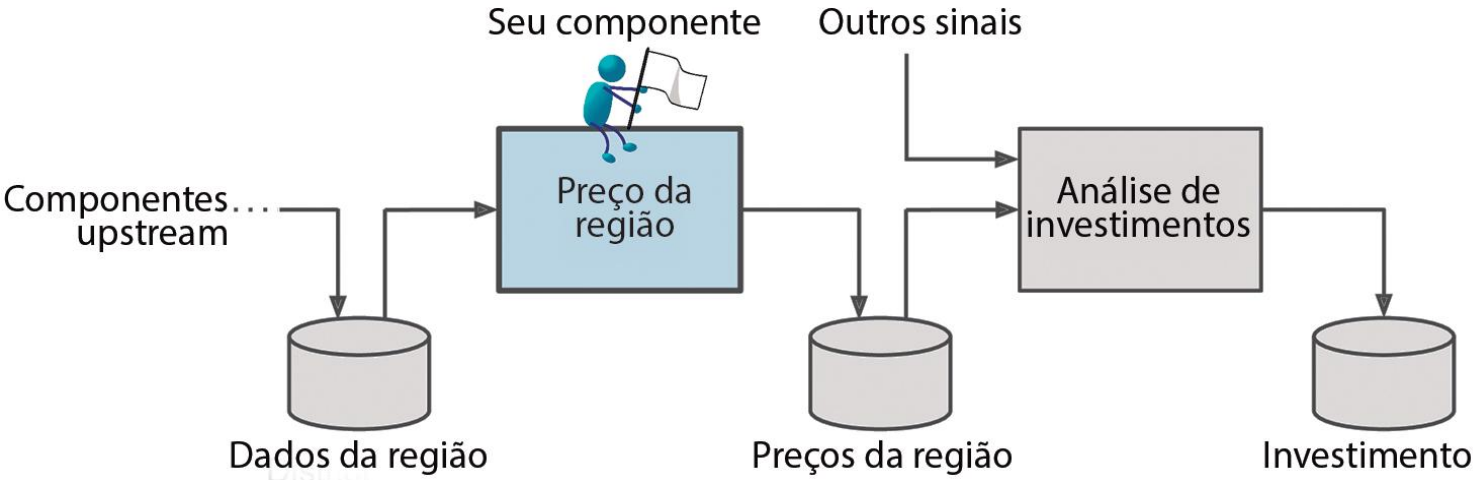
- Abordar o problema
  - Qual é exatamente o objetivo do negócio?
  - Construir um modelo não é o objetivo final.
  - Como a empresa espera usar e se beneficiar desse modelo?

Analisando o Panorama Geral

- Pipelines
  - Sequência de componentes de processamento de dados = pipeline de dados.

Analisando o Panorama Geral

- Pipelines



Analisando o Panorama Geral

- Escolha uma medida de desempenho
  - Raiz do Erro Quadrático Médio (RMSE)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

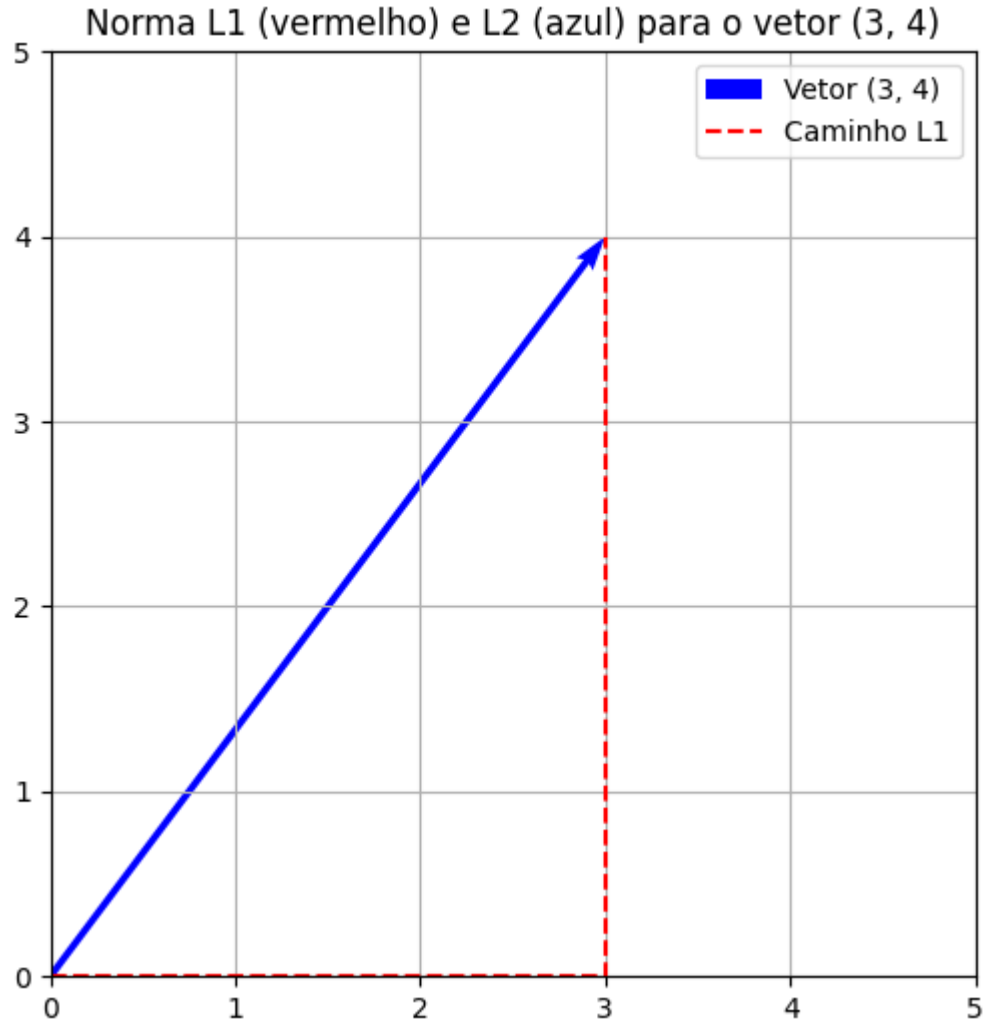
  - Erro Médio Absoluto - MAE

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right|$$

Analisando o Panorama Geral

- RMSE e MAE são formas de calcular a distância entre dois vetores:
  - o vetor das predições e o vetor dos valores-alvo.
- Diversos cálculos de distância, ou normas, são possíveis.

Norma L1 e L2 para o vetor (3,4)



Analisando o Panorama Geral

- A raiz de uma soma de quadrados (RMSE) corresponde à norma euclidiana.
- A norma ℓ2 é representada por $\|\cdot\|_2$ (ou apenas $\|\cdot\|$).
- Calculada como a raiz quadrada da soma dos quadrados de seus componentes.
- Para um vetor $v=(v_1,v_2,\ldots,v_n)$, a norma $L2$ é dada por:

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

- Cada componente contribui de forma quadrática, o que significa que valores extremos (outliers) são amplificados, pois o quadrado de um número muito grande se torna ainda maior.

Analisando o Panorama Geral

Norma ℓ1:

- Soma diretamente os valores absolutos dos componentes:

$$\|\mathbf{v}\|_{L1} = \sum |v_i|$$

- Cada componente contribui de forma **linear** para o cálculo da norma.
- Outliers (valores extremos) têm impacto **proporcional** ao seu valor absoluto, sem amplificação.

Analisando o Panorama Geral

- Diversos cálculos de distância, ou normas, são possíveis:
  - A soma de absolutos (MAE) corresponde à norma ℓ1, notado por $\| \cdot \|_1$.
    - Chamado de norma Manhattan porque calcula a distância entre dois pontos em uma cidade usando quarteirões ortogonais da cidade.
  - a norma ℓk de um vetor v que contém n elementos é definida como:

$$\| \mathbf{v} \|_k = \left(|v_0|^k + |v_1|^k + \cdots + |v_n|^k \right)^{\frac{1}{k}}$$

Analisando o Panorama Geral

- Quanto maior o índice da norma, mais ela se concentra em grandes valores e negligencia os pequenos.
- Portanto, RMSE é mais sensível aos outliers do que o MAE.
  - Quando há muitos outliers, prefere-se MAE.
- Contudo, quando os outliers são exponencialmente raros, a RMSE funciona muito bem e geralmente é a preferida.

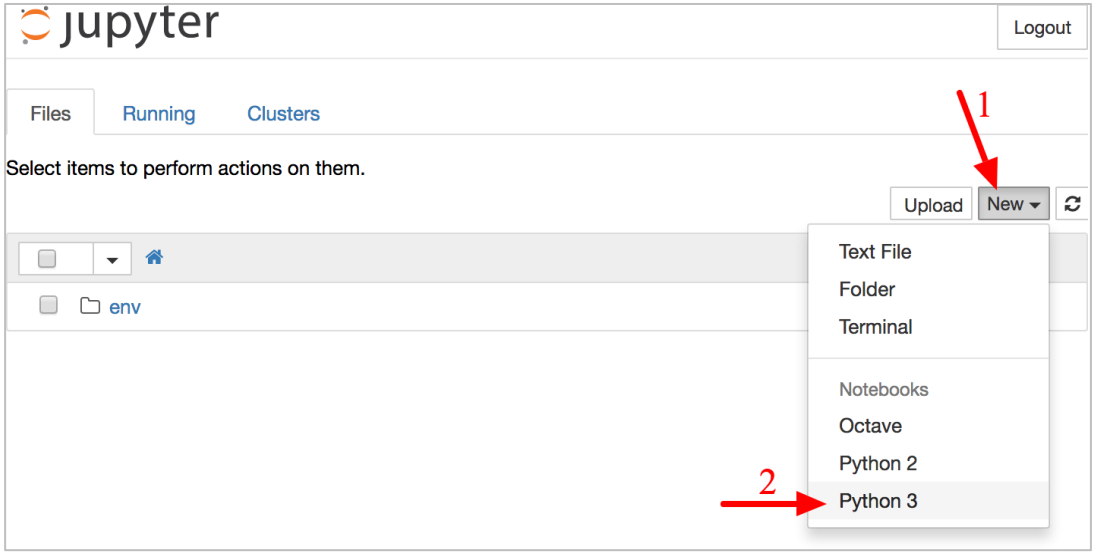
Analisando o Panorama Geral

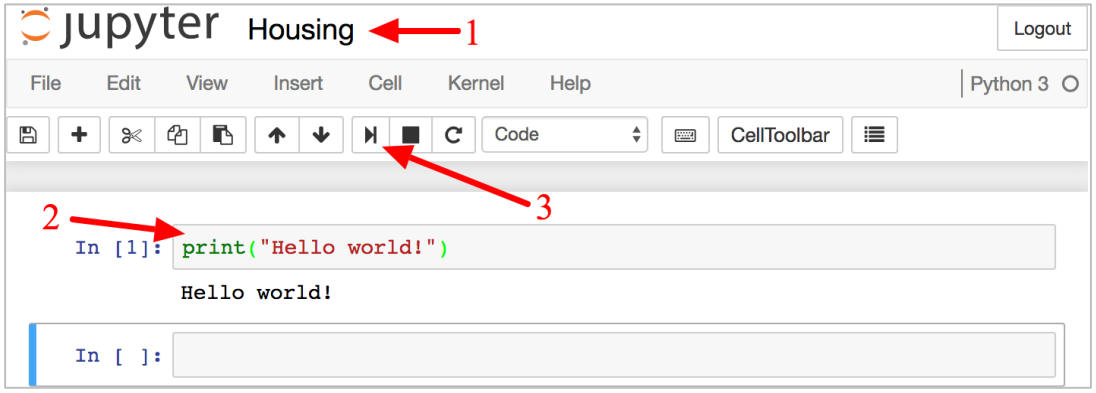
- Enumere e verifique as hipóteses
  - Exemplo: verificar se é melhor modelar o problema como classificação ou regressão.

Obtenha os dados

- Criando o workspace
  - Dependências importantes:
    - jupyter matplotlib numpy pandas scipy scikit-learn
- Criando um Ambiente Isolado
  - virtualenv (venv)
  - conda

Obtenha os dados





Obtenha os dados

- Faça o download dos dados
  - https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/housing/housing.tgz

```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

- Uma rápida olhada na estrutura dos dados

```
In [5]: housing = load_housing_data()
        housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

Obtenha os dados

- Uma rápida olhada na estrutura dos dados

```
In [6]: housing.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
>>> housing["ocean_proximity"].value_counts()
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

Obtenha os dados

- Uma rápida olhada na estrutura dos dados

```
In [8]: housing.describe()
```

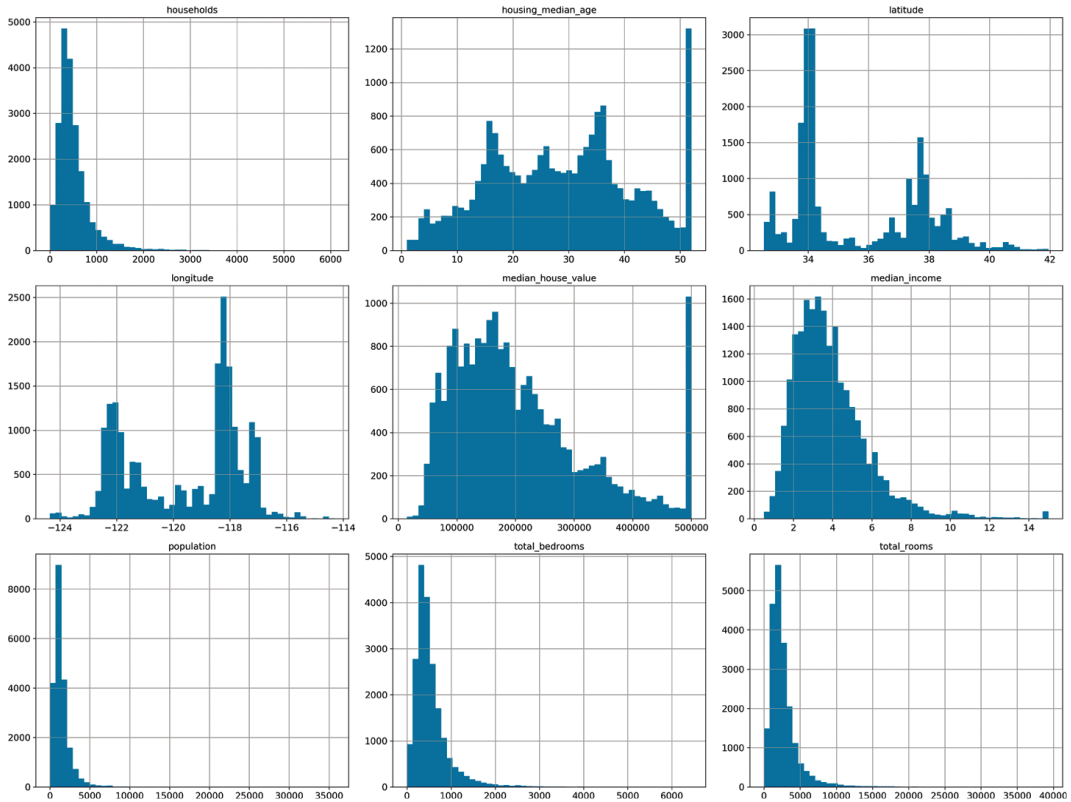
Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedr
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.38507(
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.00000(
50%	-118.490000	34.260000	29.000000	2127.000000	435.00000(
75%	-118.010000	37.710000	37.000000	3148.000000	647.00000(
max	-114.310000	41.950000	52.000000	39320.000000	6445.0000(

Obtenha os dados

- Uma rápida olhada na estrutura dos dados

```
%matplotlib inline   # only in a Jupyter notebook
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



Obtenha os dados

- Criando um conjunto de testes

```
import numpy as np

def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

```
>>> train_set, test_set = split_train_test(housing, 0.2)
>>> len(train_set)
16512
>>> len(test_set)
4128
```

Obtenha os dados

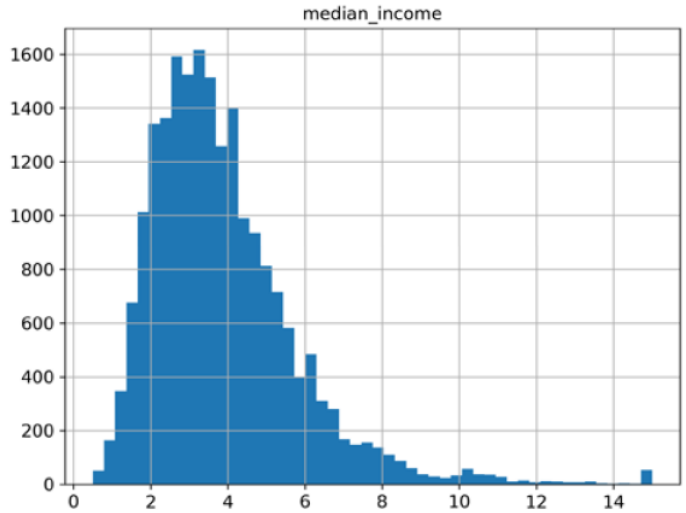
- Criando um conjunto de testes
  - Usando o scikit learn

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

Obtenha os dados

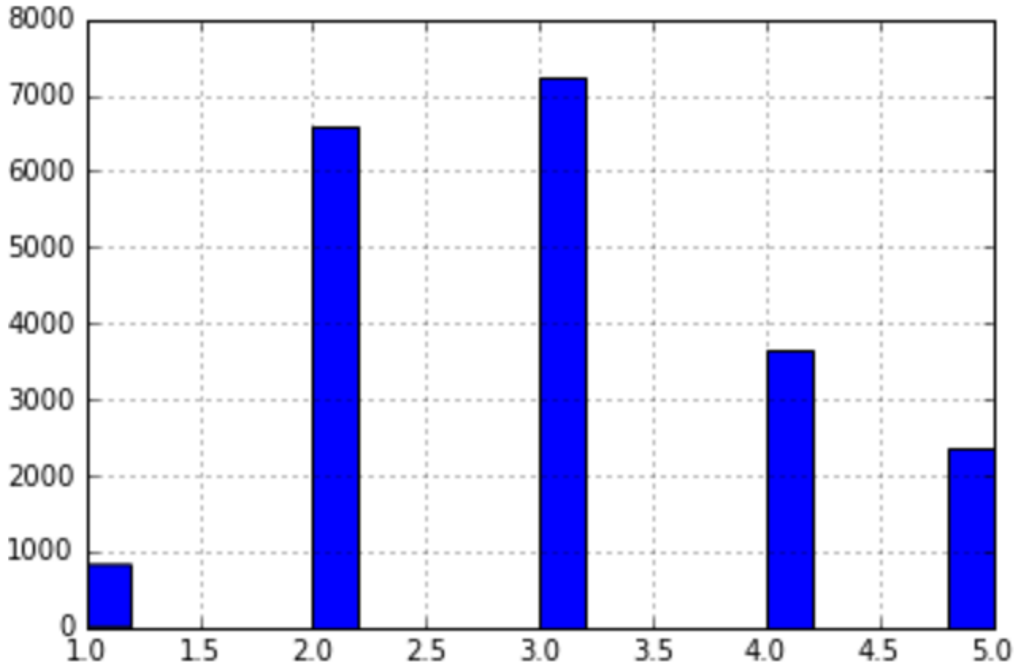
- Criando um conjunto de testes
  - É importante ter um número suficiente de instâncias para cada estrato.
    - Exemplo: renda média



Obtenha os dados

- Estratos – categoria de renda

```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```



Obtenha os dados

- Amostragem estratificada com base na categoria da renda

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

```
>>> strat_test_set["income_cat"].value_counts() / len(strat_test_set)
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

Obtenha os dados

- Amostragem estratificada x aleatória
  - O conjunto de testes gerado com amostragem estratificada tem proporções da categoria de renda quase idênticas às do conjunto completo de dados.

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

Obtenha os dados

- Remover o atributo income_cat para que os dados voltem ao estado original

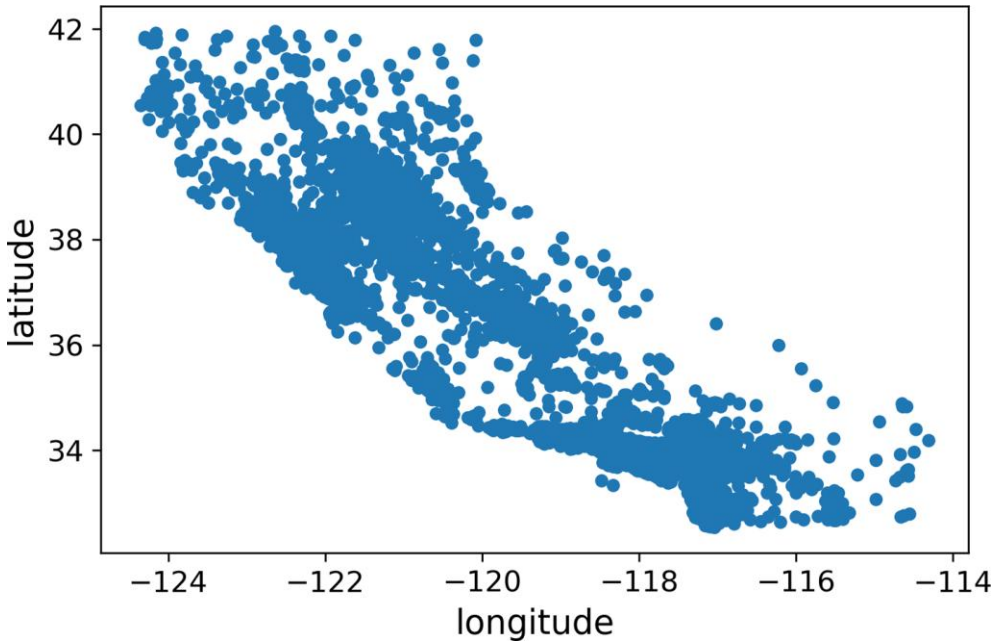
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

Explore e Visualize os Dados para Obter Informações Úteis

```
housing = strat_train_set.copy()
```

- Visualizando dados geográficos

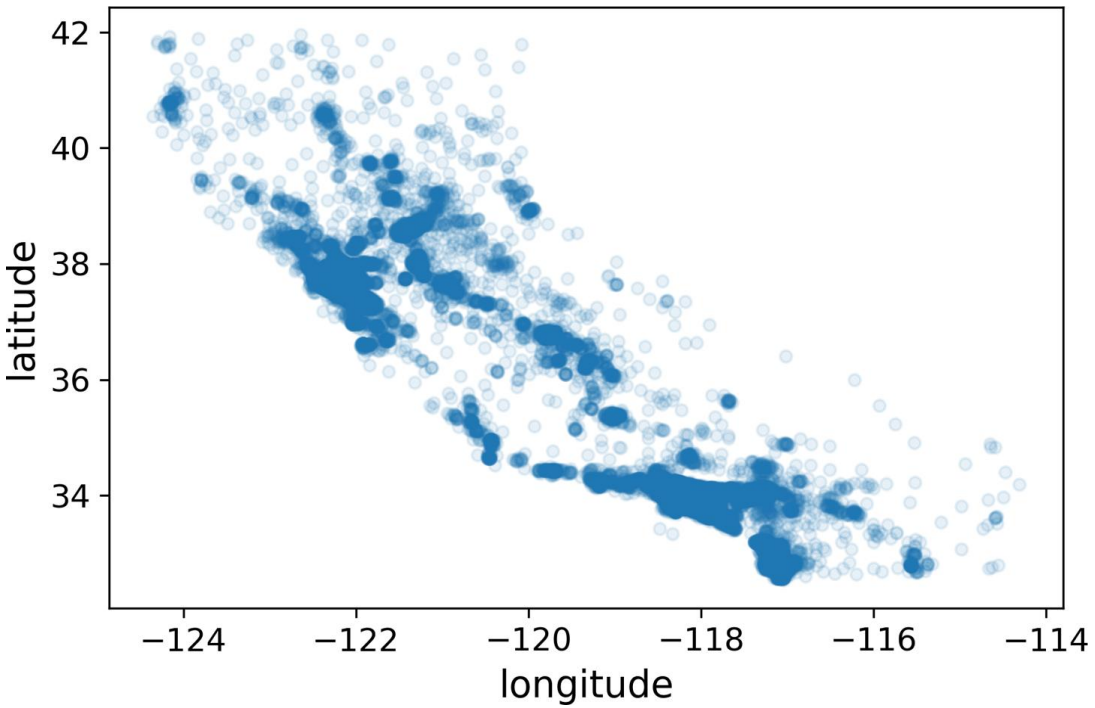
```
housing.plot(kind="scatter", x="longitude", y="latitude")
```



Explore e Visualize os Dados para Obter Informações Úteis

- Visualizando dados geográficos
  - visualização com destaque nas áreas de alta densidade

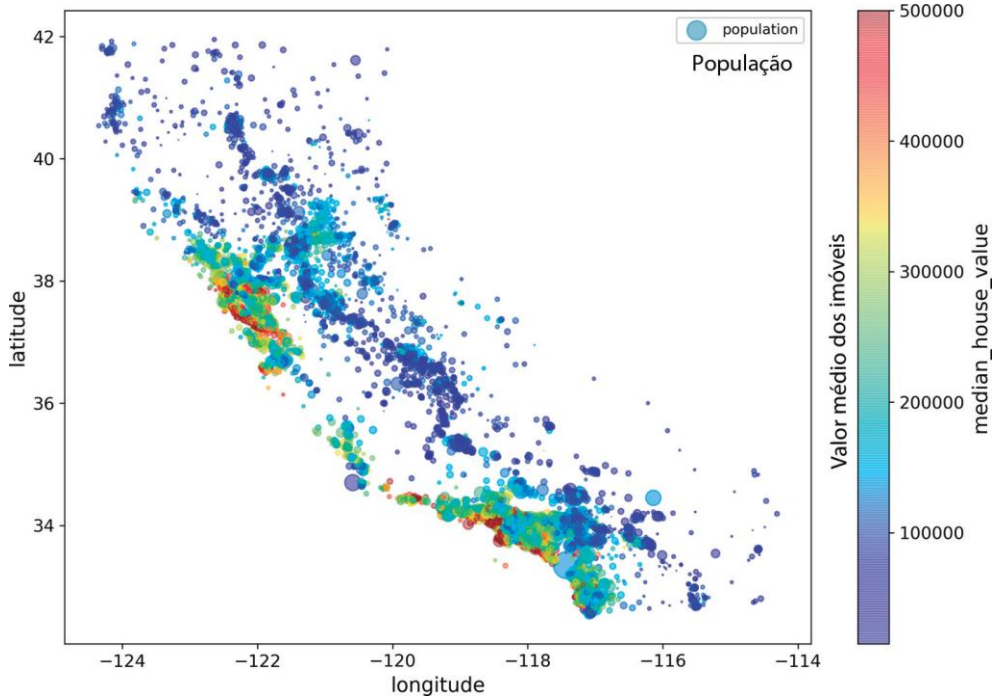
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```



Explore e Visualize os Dados para Obter Informações Úteis

- Visualizando dados geográficos

```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
    s=housing["population"]/100, label="population", figsize=(10,7),
    c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
)
plt.legend()
```



Preço médio das moradias na Califórnia

Explore e Visualize os Dados para Obter Informações Úteis

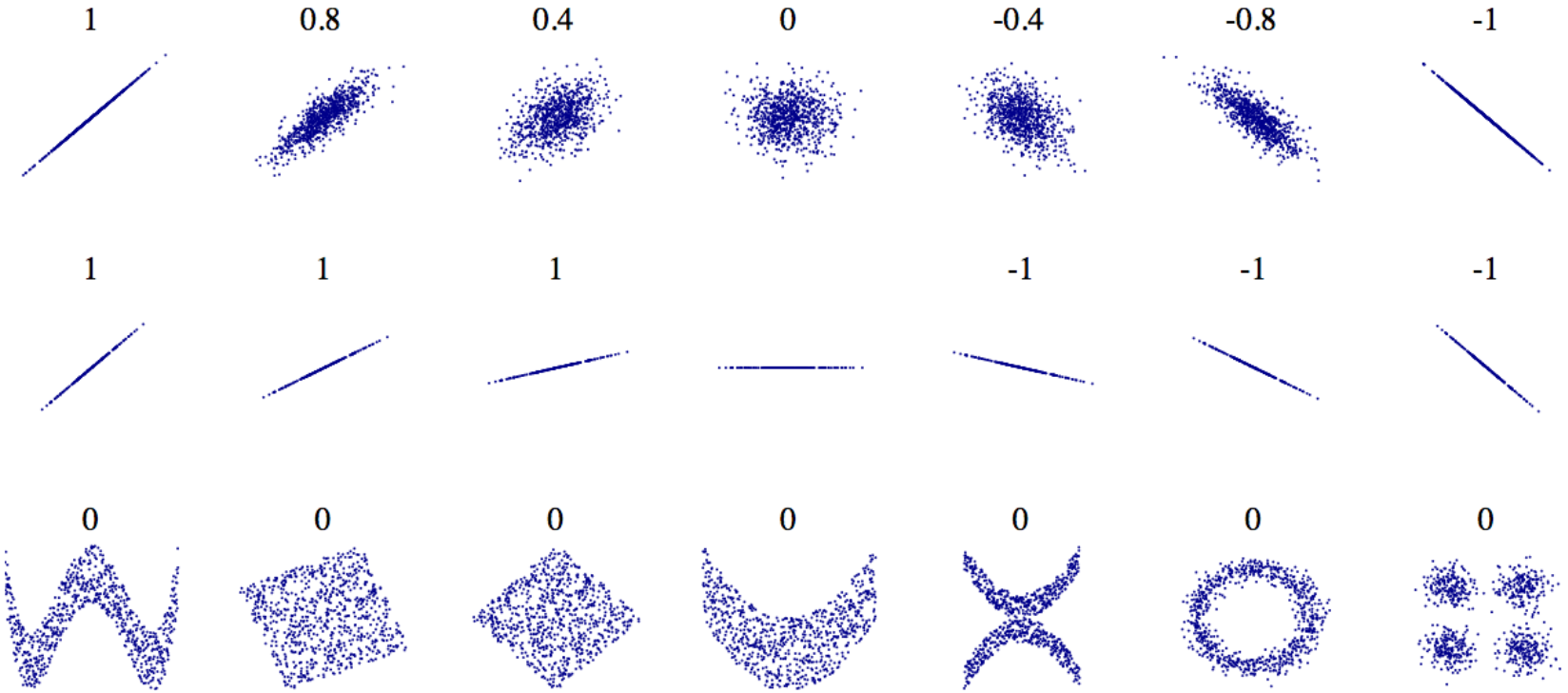
- Buscando Correlações

```
corr_matrix = housing.corr()
```

```
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.687170
total_rooms           0.135231
housing_median_age    0.114220
households            0.064702
total_bedrooms        0.047865
population           -0.026699
longitude            -0.047279
latitude             -0.142826
Name: median_house_value, dtype: float64
```

Explore e Visualize os Dados para Obter Informações Úteis

- Coeficiente de correlação padrão de vários conjuntos de dados (fonte: Wikipedia)

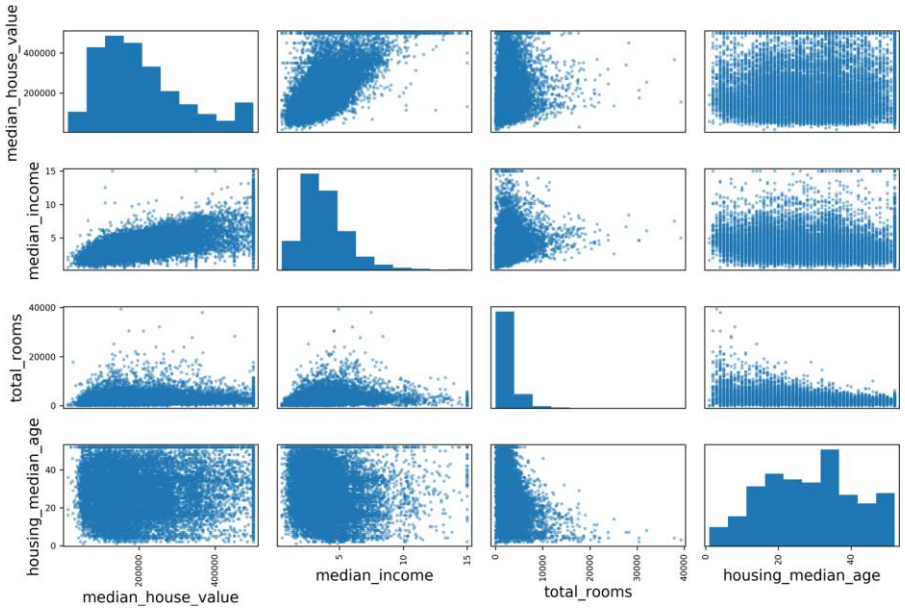


Explore e Visualize os Dados para Obter Informações Úteis

- Matriz de dispersão para plotar todos os atributos numéricos em relação a qualquer outro atributo numérico, além de um histograma de cada atributo numérico.

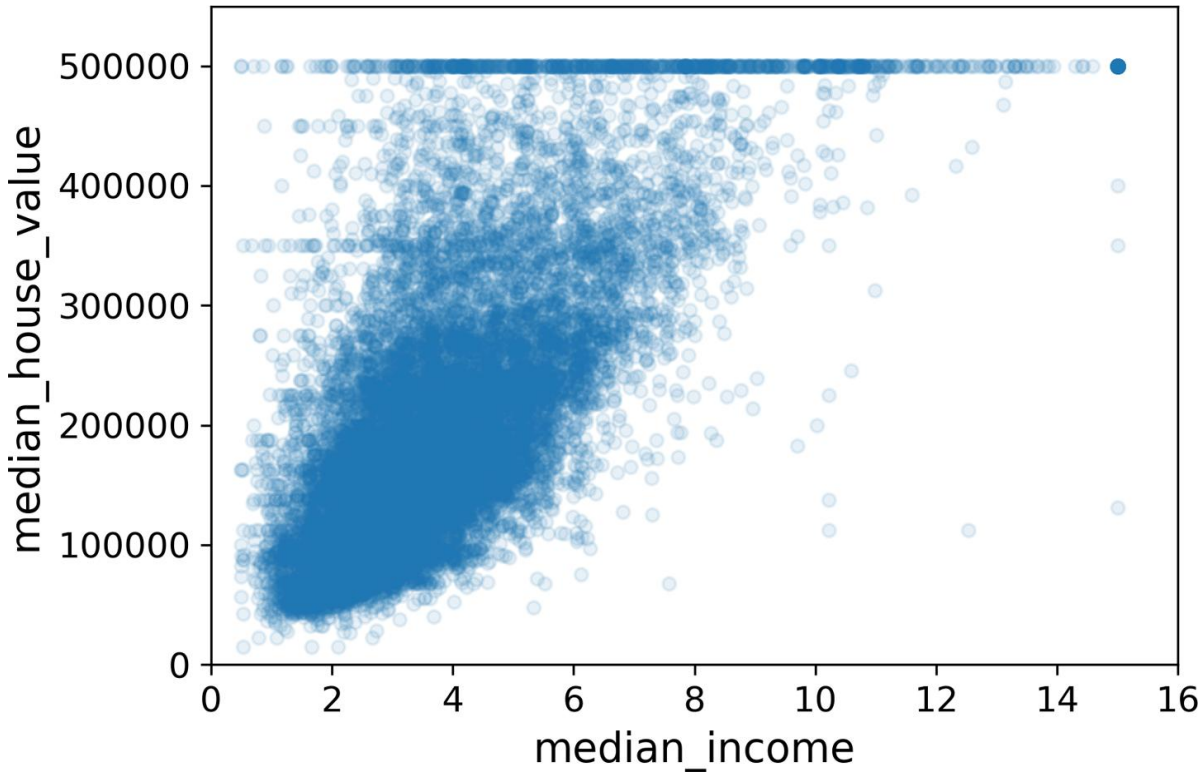
```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```



Explore e Visualize os Dados para Obter Informações Úteis

- Renda média versus valor médio dos imóveis



Explore e Visualize os Dados para Obter Informações Úteis

- Testando combinações de atributo

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

- número de cômodos por família
- número total de quartos / número de cômodos
- população por domicílio

Explore e Visualize os Dados para Obter Informações Úteis

- Testando combinações de atributo

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Separação entre preditores e rótulos
  - drop() cria uma cópia dos dados

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Limpando os dados
  - A maioria dos algoritmos de AM não funciona com características ausentes.
  - Opções ao se deparar com valores faltantes no atributo total_bedrooms:
    1. Abrir mão das regiões correspondentes.
    2. Abrir mão de todo o atributo.
    3. Definir os valores para algum valor (zero, a média, a mediana etc.).

```
housing.dropna(subset=["total_bedrooms"])    # option 1
housing.drop("total_bedrooms", axis=1)       # option 2
median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Limpando os dados
  - Scikit Learn tem a classe SimpleImputer para lidar com valores faltantes.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

housing_num = housing.drop("ocean_proximity", axis=1)

imputer.fit(housing_num)

>>> imputer.statistics_
array([ -118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])
>>> housing_num.median().values
array([ -118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])

X = imputer.transform(housing_num)

housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- O design da biblioteca Scikit-Learn
  - Princípios de design usados no Scikit:
    - Consistência - todos os objetos compartilham uma interface simples e consistente:
      - Estimadores - Qualquer objeto capaz de estimar alguns parâmetros com base em um conjunto de dados se chama de estimador.
        - Exemplo: SimpleImputer
        - A estimativa em si é realizada pelo método fit().
      - Transformadores - Alguns estimadores também podem transformar um conjunto de dados e se chamam transformadores.
        - A transformação é realizada pelo método transform(), que transforma o conjunto de dados.
          - Ele retorna o conjunto de dados transformados.
        - Todos os transformadores também têm um método fit_transform()
          - Equivalente a chamar o fit() e depois o transform().
          - Algumas vezes o fit_transform() é otimizado e roda muito mais rápido.

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- O design da biblioteca Scikit-Learn
  - Princípios de design usados no Scikit:
    - Preditores
      - Alguns estimadores são capazes de realizar previsões.
      - Eles se chamam preditores.
      - Exemplo: LinearRegression
      - Um preditor tem um método predict()
        - Pega um conjunto de dados de instâncias novas e retorna um conjunto de dados de predições correspondentes.
      - Tem um método score()
        - Computa a qualidade das predições, levando em conta um conjunto de teste (e os rótulos correspondentes (aprendizado supervisionado).
    - Inspeção
      - Todos os hiperparâmetros do estimador são diretamente acessados por meio de variáveis das instâncias públicas (Exemplo: imputer.strategy)
      - Todos os parâmetros aprendidos do estimador também são acessados por meio das variáveis das instâncias públicas seguido de "_" (Exemplo: imputer.statistics_).

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- O design da biblioteca Scikit-Learn
  - Princípios de design usados no Scikit:
    - Não proliferação de classes
      - Os conjuntos de dados são representados como matrizes NumPy ou matrizes esparsas SciPy
      - Hiperparâmetros são apenas strings ou números do Python.
    - Composição
      - Blocos de construção são reutilizados tanto quanto possível.
      - Exemplo: estimador Pipeline a partir de uma sequência arbitrária de transformadores seguida de um estimador final.
    - Padrões plausíveis
      - Valores-padrão aceitáveis para a maioria dos parâmetros, facilitando a criação rápida de um sistema de trabalho de referência.

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Manipulando texto e atributos categóricos
  - A maioria dos algoritmos de AM prefere trabalhar com números.

```
>>> housing_cat = housing[["ocean_proximity"]]
>>> housing_cat.head(10)
       ocean_proximity
17606        <1H OCEAN
18632        <1H OCEAN
14650       NEAR OCEAN
3230            INLAND
3555         <1H OCEAN
19480           INLAND
8879         <1H OCEAN
13685           INLAND
4937         <1H OCEAN
4861         <1H OCEAN
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Manipulando texto e atributos categóricos
  - A maioria dos algoritmos de AM prefere trabalhar com números.

```
>>> from sklearn.preprocessing import OrdinalEncoder
>>> ordinal_encoder = OrdinalEncoder()
>>> housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
>>> housing_cat_encoded[:10]
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])

>>> ordinal_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Manipulando texto e atributos categóricos
  - Um dos problemas com essa representação é que os algoritmos de AM assumem que dois valores próximos são mais semelhantes que dois valores distantes.
  - Isso pode ser bom em alguns casos
    - Exemplo: categorias ordenadas como “ruim”, “médio”, “bom” e “excelente”.
  - Uma solução comum é criar um atributo binário por categoria.
    - **One Hot Encoding**

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> cat_encoder = OneHotEncoder()
>>> housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
>>> housing_cat_1hot
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
  with 16512 stored elements in Compressed Sparse Row format>
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Manipulando texto e atributos categóricos

```
>>> housing_cat_1hot.toarray()
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

```
>>> cat_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Se um atributo categórico tiver um grande número de categorias possíveis, a codificação one-hot resultará em um grande número de características.
- Isso pode retardar o treinamento e prejudicar o desempenho.
- Alternativas:
  - Substituir a entrada categórica por características numéricas úteis relacionadas às categorias.
    - Ex:
      - substituir a categoria ocean_proximity pelo atributo binário “ao nível do mar”.
      - Um código de país pode ser substituído pela população e pelo PIB per capita do país.
  - Substituir categoria por um vetor de baixa dimensão e capaz de aprender, chamado embedding.
    - A representação de cada categoria seria aprendida durante o treinamento (aprendizado por representação – Cap. 13 e 17).

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Se um atributo categórico tiver um grande número de categorias possíveis, a codificação one-hot resultará em um grande número de características.
- Isso pode retardar o treinamento e prejudicar o desempenho.

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Alternativas à codificação one-hot:
  - Substituir a entrada categórica por características numéricas úteis relacionadas às categorias.
    - Ex:
      - substituir a categoria ocean_proximity pelo atributo binário “ao nível do mar”.
      - Um código de país pode ser substituído pela população e pelo PIB per capita do país.
  - Substituir categoria por um vetor de baixa dimensão e capaz de aprender, chamado embedding.
    - A representação de cada categoria seria aprendida durante o treinamento (aprendizado por representação – Cap. 13 e 17).

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Customize os Transformadores
  - Escreva seus próprios para tarefas como operações de limpeza customizadas ou combinar atributos específicos.
    - Crie uma classe e implemente os três métodos: **fit( )**, **transform( )** e **fit_transform( )**.

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Customize os Transformadores

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Escalonamento das Características
  - Os algoritmos de ML não funcionam bem quando atributos numéricos de entrada têm escalas muito diferentes.
  - Ex: o número total de cômodos varia de 6 a 39.320, ao passo que a renda média varia apenas de 0 a 15.
  - Geralmente não é necessário escalonar os valores-alvo (rótulos).
  - Há duas formas comuns de todos os atributos terem a mesma escala: escalonamento min-max e padronização.
    - Escalonamento min-max (normalização)
      - os valores são deslocados e reescalonados de modo que acabam variando de 0 a 1.
      - No Scikit: MinMaxScaler.

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Escalonamento das Características
  - Padronização
    - Subtrair o valor médio (os valores padronizados sempre têm média zero) e, dividir pelo desvio-padrão para que a distribuição resultante tenha variação de unidade.
    - A padronização não vincula valores a um intervalo específico, o que pode ser um problema para alguns algoritmos (Ex: as redes neurais geralmente esperam um valor de entrada que varia de 0 a 1).
    - No entanto, a padronização não é tão afetada pelos outliers.
      - Ex: suponha que uma região tenha uma renda média igual a 100 (por engano). O escalonamento min-max condensaria todos os outros valores de 0-15 para 0-0,15, enquanto a padronização não seria muito afetada.
    - No Scikit: StandardScaler.
    - **IMPORTANTE:**
      - Em todas as transformações:
        - **Ajustar os escalonamentos apenas aos dados de treinamento**, não ao conjunto de dados completo (incluindo o conjunto de testes).
        - Só então você poderá usá-los para transformar o conjunto de treinamento e o conjunto de testes (e os dados novos).

Escalonamento das Características

$$x^{(i)}_{std} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

standardization

$$x^{(i)}_{norm} = \frac{x^{(i)} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}}$$

min-max scaling ("normalization")

	input	standardized	normalized
0	0	-1.46385	0.0
1	1	-0.87831	0.2
2	2	-0.29277	0.4
3	3	0.29277	0.6
4	4	0.87831	0.8
5	5	1.46385	1.0

Normalization

```
np.random.seed(0)
x = np.random.rand(20)
x = (x * 100).round(2)
x = np.resize(x, (20, 1))
```

```
[[ 54.88]
 [ 71.52]
 [ 60.28]
 [ 54.49]
 [ 42.37]
 [ 64.59]
 [ 43.76]
 [ 89.18]
 [ 96.37]
 [ 38.34]
 [ 79.17]
 [ 52.89]
 [ 56.8 ]
 [ 92.56]
 [  7.1 ]
 [  8.71]
 [  2.02]
 [ 83.26]
 [ 77.82]
 [ 87.  ]]
```

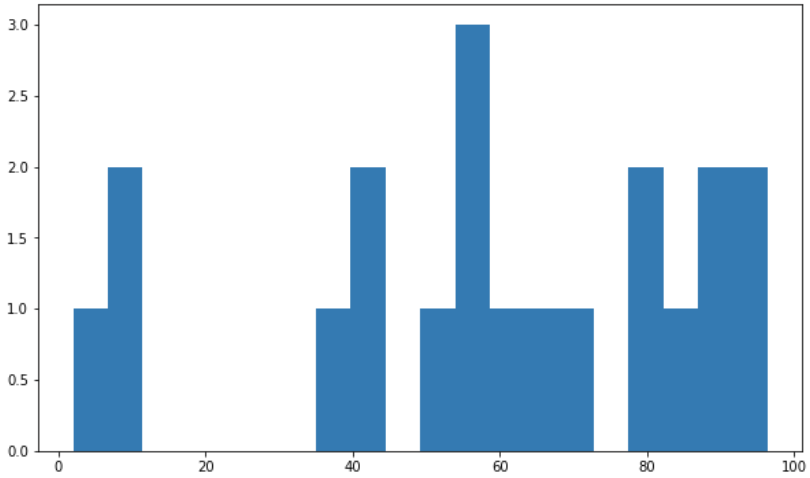
$$x_{norm}^{(i)} = \frac{x^{(i)} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}}$$

```
def normalize(X):
    X_norm = np.copy(X)
    n_cols = X.shape[1]
    for i in range(n_cols):
        X_norm[:, i] = (X[:, i] - np.min(X[:, i])) /
                       (np.max(X[:, i]) - np.min(X[:, i]))
    return X_norm
```

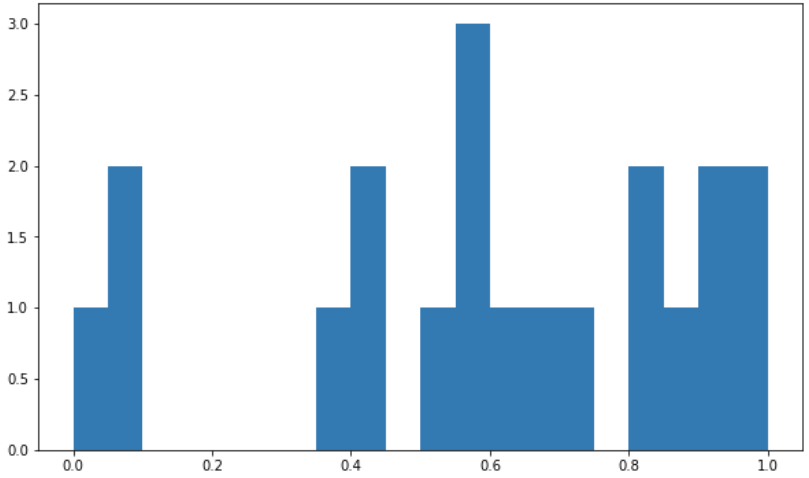
Normalization

```
x_norm = normalize(x)
```

```
plt.hist(x, bins=20)
```




```
x
---
mean: 58.16,
std: 27.59,
min: 2.02,
max: 96.37
```




```
x_norm
---
mean: 0.59,
std: 0.29,
min: 0.0,
max: 1.0
```

```
[[ 0.56025437],
 [ 0.73661897],
 [ 0.61748808],
 [ 0.55612083],
 [ 0.42766296],
 [ 0.66316905],
 [ 0.44239534],
 [ 0.92379438],
 [ 1.        ],
 [ 0.38494966],
 [ 0.81770005],
 [ 0.53916269],
 [ 0.58060413],
 [ 0.95961844],
 [ 0.05384208],
 [ 0.0709062 ],
 [ 0.        ],
 [ 0.86104928],
 [ 0.80339163],
 [ 0.90068892]]
```

Standardization

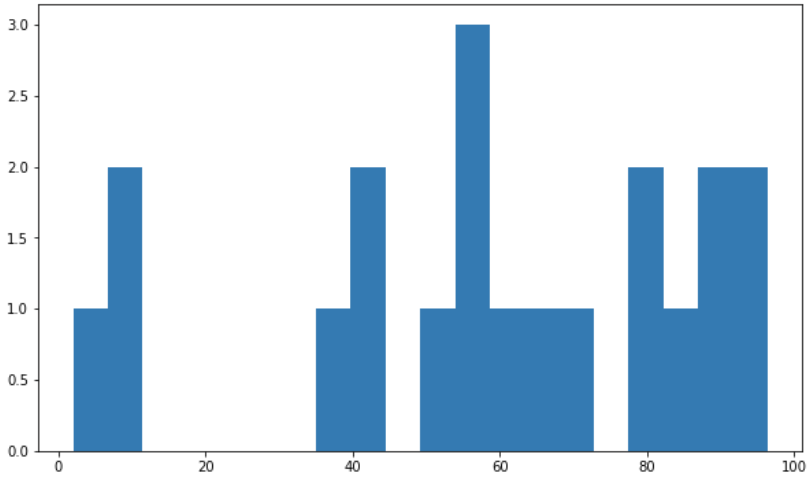
$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

```
def standardize(X):
    X_std = np.copy(X)
    n_cols = X.shape[1]
    for i in range(n_cols):
        X_std[:, i] = (X[:, i] - np.mean(X[:, i])) /
                      np.std(X[:, i])
    return X_std
```

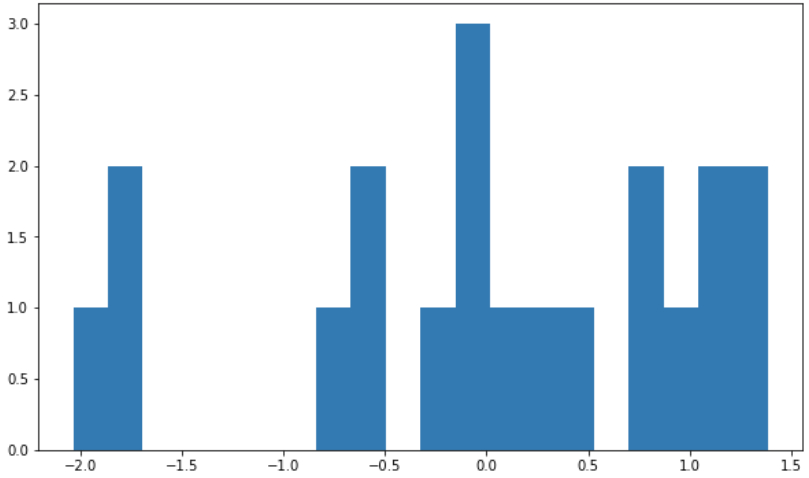
Standardization

```
x_std = standardize(x)
```

```
plt.hist(x, bins=20)
```




```
x
---
mean: 58.16,
std: 27.59,
min: 2.02,
max: 96.37
```




```
x_std
---
mean: 0.0,
std: 1.0,
min: -2.03,
max: 1.38
```

```
[[-0.11870903],
 [ 0.48434953],
 [ 0.07699507],
 [-0.13284322],
 [-0.5720902 ],
 [ 0.23319593],
 [-0.52171451],
 [ 1.12437442],
 [ 1.38495081],
 [-0.71814345],
 [ 0.761597  ],
 [-0.19082962],
 [-0.04912535],
 [ 1.24687069],
 [-1.85032791],
 [-1.79197909],
 [-2.03443473],
 [ 0.90982474],
 [ 0.71267098],
 [ 1.04536795]]
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Transformação de Pipelines
  - Existem muitas etapas de transformação de dados que precisam ser executadas na ordem correta.
  - A classe **Pipeline** ajuda com essas sequências de transformações.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Transformação de Pipelines
  - É conveniente ter um único transformador para lidar com todas as colunas, aplicando as transformações adequadas a cada coluna.
  - No Scikit: ColumnTransformer - funciona bem com os DataFrames do Pandas.

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Transformação de Pipelines
  - O OneHotEncode retorna uma matriz esparsa, ao passo que o num_pipeline retorna uma matriz densa.

Prepare os Dados para Algoritmos de Aprendizado de Máquina

- Transformação de Pipelines
  - Em vez de usar um transformador, pode-se especificar a string "drop" caso queira que as colunas sejam dropadas (default), ou pode especificar um "pass through" se desejar que as colunas permaneçam inalteradas.

Escolha e treine um modelo

- Treinando e avaliando o conjunto de treinamento

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

```
>>> some_data = housing.iloc[:5]
>>> some_labels = housing_labels.iloc[:5]
>>> some_data_prepared = full_pipeline.transform(some_data)
>>> print("Predictions:", lin_reg.predict(some_data_prepared))
Predictions: [ 210644.6045  317768.8069  210956.4333  59218.9888  189747.5584]
>>> print("Labels:", list(some_labels))
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

```
>>> from sklearn.metrics import mean_squared_error
>>> housing_predictions = lin_reg.predict(housing_prepared)
>>> lin_mse = mean_squared_error(housing_labels, housing_predictions)
>>> lin_rmse = np.sqrt(lin_mse)
>>> lin_rmse
68628.19819848922
```

Escolha e treine um modelo

- Treinando e avaliando o conjunto de treinamento
  - DecisionTreeRegressor – poderoso: capaz de identificar relacionamentos não lineares complexos nos dados.

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

```
>>> housing_predictions = tree_reg.predict(housing_prepared)
>>> tree_mse = mean_squared_error(housing_labels, housing_predictions)
>>> tree_rmse = np.sqrt(tree_mse)
>>> tree_rmse
0.0
```

  - É provável que tenha ocorrido overfitting!!!

Escolha e treine um modelo

- Avaliando melhor com a validação cruzada
  - Função train_test_split() divide o conjunto de treino em um conjunto menor e em um conjunto de validação.
  - Depois, treina-se os modelos com esse conjunto menor e avalia-os em relação ao conjunto de validação.

Escolha e treine um modelo

- Avaliando melhor com a validação cruzada
  - Outra alternativa é usar o método k-fold de validação cruzada:
    - Dividir aleatoriamente o conjunto de treinamento em *k* subconjuntos distintos chamados folds.
    - Treinar e avaliar o modelo *k* vezes, escolhendo sempre um fold diferente para avaliação e treinamento dos outros *(k-1)* folds.
    - O resultado é um array que contém *k* classificações de avaliação.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

Escolha e treine um modelo

- Avaliando melhor com a validação cruzada
  - Os recursos de validação cruzada do Scikit esperam uma função de utilidade (quanto maior, melhor) do que uma função de custo (quanto menor, melhor).
  - Portanto, a função score é de fato oposta à MSE (ou seja, um valor negativo).

Escolha e treine um modelo

- Avaliando melhor com a validação cruzada

```
>>> def display_scores(scores):
...     print("Scores:", scores)
...     print("Mean:", scores.mean())
...     print("Standard deviation:", scores.std())
...
>>> display_scores(tree_rmse_scores)
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

O modelo de árvore de decisão está sobreajustando tanto que acaba sendo pior do que o modelo de regressão linear.

```
>>> lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
...                              scoring="neg_mean_squared_error", cv=10)
...
>>> lin_rmse_scores = np.sqrt(-lin_scores)
>>> display_scores(lin_rmse_scores)
Scores: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798348
```

Escolha e treine um modelo

- Avaliando melhor com a validação cruzada

```
>>> def display_scores(scores):
...     print("Scores:", scores)
...     print("Mean:", scores.mean())
...     print("Standard deviation:", scores.std())
...
>>> display_scores(tree_rmse_scores)
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> forest_reg = RandomForestRegressor()
>>> forest_reg.fit(housing_prepared, housing_labels)
>>> [...]
>>> forest_rmse
18603.515021376355
>>> display_scores(forest_rmse_scores)
Scores: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

Esse modelo é muito melhor.
Florestas aleatórias parecem muito boas.
No entanto, o score no conjunto de treino é muito menor do que nos conjuntos de validação.
Significa que o modelo ainda está se sobreajustando ao conjunto de treinamento.

Escolha e treine um modelo

- Avaliando melhor com a validação cruzada
  - Possíveis soluções para sobreajustes:
    - simplificar o modelo
    - restringi-lo (ou seja, regularizá-lo)
    - obter mais dados de treinamento.
  - Deve-se testar muitos outros modelos de diversas categorias de algoritmos de AM

Aperfeiçoe Seu Modelo

- Grid Search
  - Scikit Learn: GridSearchCV
    - Defina os hiperparâmetros e valores deseja que ele teste, e ele avaliará todas as combinações de valores possíveis por meio de validação cruzada.

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(housing_prepared, housing_labels)
```

Aperfeiçoe Seu Modelo

- Grid Search
  - Obter a melhor combinação de hiper-parâmetros:

```
>>> grid_search.best_params_
{'max_features': 8, 'n_estimators': 30}
```

  - Como 8 e 30 são os valores máximos que foram avaliados, você deve tentar pesquisar novamente com valores mais altos, pois o score pode melhorar ainda mais.
  - O melhor modelo pode ser obtido assim:

```
>>> grid_search.best_estimator_
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=8, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=30, n_jobs=None, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

Aperfeiçoe Seu Modelo

- Grid Search
  - GridSearchCV com refit = True (que é o padrão), depois de identificar o melhor estimador usando a validação cruzada, ele o restringe em todo o conjunto de treinamento.
  - Ao fornecer mais dados, provavelmente, melhorará o desempenho do modelo.

Aperfeiçoe Seu Modelo

- Os **scores de avaliação** também estão disponíveis:

```
>>> cvres = grid_search.cv_results_
>>> for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
...     print(np.sqrt(-mean_score), params)
...
63669.05791727153 {'max_features': 2, 'n_estimators': 3}
55627.16171305252 {'max_features': 2, 'n_estimators': 10}
53384.57867637289 {'max_features': 2, 'n_estimators': 30}
60965.99185930139 {'max_features': 4, 'n_estimators': 3}
52740.98248528835 {'max_features': 4, 'n_estimators': 10}
50377.344409590376 {'max_features': 4, 'n_estimators': 30}
58663.84733372485 {'max_features': 6, 'n_estimators': 3}
52006.15355973719 {'max_features': 6, 'n_estimators': 10}
50146.465964159885 {'max_features': 6, 'n_estimators': 30}
57869.25504027614 {'max_features': 8, 'n_estimators': 3}
51711.09443660957 {'max_features': 8, 'n_estimators': 10}
49682.25345942335 {'max_features': 8, 'n_estimators': 30}
62895.088889905004 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.14484390074 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.399594730654 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52725.01091081235 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.612956065226 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.51445842374 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Aperfeiçoe Seu Modelo

- Randomized Search
  - Grid search é bom quando há poucas combinações.
  - Mas, quando o espaço de pesquisa de hiperparâmetro é grande, é preferível usar o **RandomizedSearchCV**.
  - Em vez de testar todas as combinações possíveis, ela avalia um determinado número de combinações aleatórias.
  - Basta definir o número de iterações, que você terá mais controle sobre o custo computacional que deseja alocar para a pesquisa por hiperparâmetro.

Aperfeiçoe Seu Modelo

- Métodos ensemble
  - O agrupamento (ou “ensemble”) geralmente terá um desempenho superior em relação ao melhor modelo individual.
    - Ex: As florestas aleatórias funcionam melhor do que as árvores de decisão, sobretudo se os modelos individuais tiverem tipos muito diferentes de erros.

Aperfeiçoe Seu Modelo

- Analise os melhores modelos e seus erros
  - Você obtém informações reveladoras sobre o problema inspecionando os melhores modelos.
  - Ex: o RandomForestRegressor pode indicar a importância relativa de cada atributo para realizar predições precisas:

```
>>> feature_importances = grid_search.best_estimator_.feature_importances_
>>> feature_importances
array([7.33442355e-02, 6.29090705e-02, 4.11437985e-02, 1.46726854e-02,
       1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,
       5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,
       1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03])
```

Aperfeiçoe Seu Modelo

- Analise os melhores modelos e seus erros
  - Scores de importância ao lado dos nomes de atributos correspondentes:

```
>>> extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
>>> cat_encoder = full_pipeline.named_transformers_["cat"]
>>> cat_one_hot_attribs = list(cat_encoder.categories_[0])
>>> attributes = num_attribs + extra_attribs + cat_one_hot_attribs
>>> sorted(zip(feature_importances, attributes), reverse=True)
[(0.3661589806181342, 'median_income'),
 (0.1647809935615905, 'INLAND'),
 (0.10879295677551573, 'pop_per_hhold'),
 (0.07334423551601242, 'longitude'),
 (0.0629090704826203, 'latitude'),
 (0.05641917918195401, 'rooms_per_hhold'),
 (0.05335107734767581, 'bedrooms_per_room'),
 (0.041143798478729635, 'housing_median_age'),
 (0.014874280890402767, 'population'),
 (0.014672685420543237, 'total_rooms'),
 (0.014257599323407807, 'households'),
 (0.014106483453584102, 'total_bedrooms'),
 (0.010311488326303787, '<1H OCEAN'),
 (0.002856474637320158, 'NEAR OCEAN'),
 (0.00196041559947807, 'NEAR BAY'),
 (6.028038672736599e-05, 'ISLAND')]
```

Aperfeiçoe Seu Modelo

- Analise os melhores modelos e seus erros
  - Com essa informação, pode-se descartar algumas das características menos úteis.
    - Ex: Apenas uma categoria da ocean_proximity é de fato útil; logo, você pode tentar descartar as outras.

Aperfeiçoe Seu Modelo

- Avalie seu sistema no conjunto de testes
  - Obter os preditores e os rótulos do seu conjunto de testes;
  - Execute seu full_pipeline para transformar os dados (chame o transform(), não o fit_transform());
  - Avalie o modelo final no conjunto de testes:

```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)

final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)   # => evaluates to 47,730.2
```

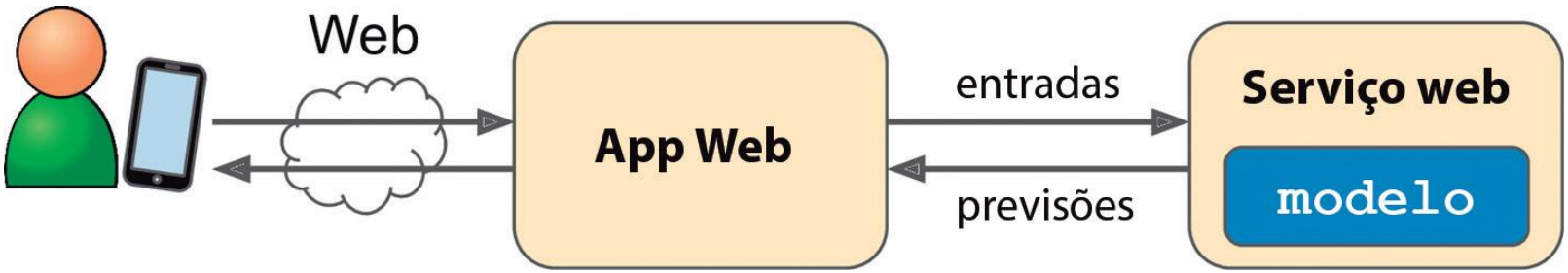
Aperfeiçoe Seu Modelo

- Avalie seu sistema no conjunto de testes
  - Melhorando a estimativa pontual do erro de generalização:
    - Calcular um intervalo de confiança de 95% para o erro de generalização, usando scipy.stats.t.interval():

```
>>> from scipy import stats
>>> confidence = 0.95
>>> squared_errors = (final_predictions - y_test) ** 2
>>> np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
...                          loc=squared_errors.mean(),
...                          scale=stats.sem(squared_errors)))
...
array([45685.10470776, 49691.25001878])
```

Implemente, Monitore e Faça a Manutenção de Seu Sistema

- Implementar o modelo em seu ambiente de produção.
  - salvar o modelo treinado.
  - Fazer upload do modelo treinado em seu ambiente de produção.
  - Usá-lo para realizar predições chamando o método predict().
  - Encapsular o modelo em um serviço web dedicado que seu aplicativo web pode consultar por meio de uma REST API.



Implemente, Monitore e Faça a Manutenção de Seu Sistema

- Escrever um código de monitoramento para verificar o desempenho em tempo real do seu sistema em intervalos regulares e acionar alertas quando ele cair.
- Atualizar seus conjuntos de dados e treinar seu modelo regularmente.

Teste

- Familiarize-se com o processo
  - Selecione um conjunto de dados de seu interesse.
  - Etapa de preparação de dados:
    - Desenvolva ferramentas de monitoramento;
    - Configure pipelines de avaliação humana;
    - Automatize o treinamento regular de modelos.
  - Teste todo o processo do começo ao fim.

Obrigado!

Dúvidas, comentários, sugestões?

Regis Pires Magalhães
regismagalhaes@ufc.br

UFC