Projeto de aprendizado de máquina de ponta a ponta

Vamos desenvolver um projeto completo de aprendizado de máquina, do início ao fim, para entendermos a necessidade de vários fundamentos teóricos. O principal objetivo aqui é conhecer o processo geral, sem necessidade de domínio do código por enquanto.

Etapas do projeto de Aprendizado de Máquina:

1. **Análise do Problema**

- Entendimento do Problema

2. **Análise de Dados**

- Obtenção dos Dados
- Exploração dos Dados
- Preparação dos Dados

3. **Construção do Modelo**

- Seleção do Modelo
- Treinamento do Modelo

4. **Avaliação do Modelo**

- Avaliação do Modelo
- Ajuste do Modelo

5. **Implantação e Manutenção do Sistema**

- Apresentação da Solução
- Lançamento do Sistema
- Monitoração do Sistema
- Manutenção do Sistema

ETAPA 01 - Análise do Problema

A) Entendimento do Problema

A primeira pergunta a fazer ao seu chefe é qual é exatamente o objetivo do negócio.

Construir um modelo provavelmente não é o objetivo final.

Como a empresa espera usar e se beneficiar desse modelo?

Conhecer o objetivo é importante porque determinará como você enquadra o problema, quais algoritmos selecionará, qual medida de desempenho usará para avaliar seu modelo e quanto esforço gastará para ajustá-lo.

Seu chefe responde que a saída do seu modelo (uma previsão do preço médio da habitação de um distrito) será enviada para outro sistema de Machine Learning (veja a Figura), juntamente com muitos outros sinais.

Esse sistema de fluxo determinará se vale a pena investir em uma determinada área ou não. Acertar isso é fundamental, pois afeta diretamente a receita.

Bem-vindo à Machine Learning Housing Corp.!

Objetivo: prever os valores médios das casas nos distritos da Califórnia, considerando várias características desses distritos.

Tarefa: construir um modelo de preços do setor imobiliário utilizando os dados do censo da Califórnia. Seu modelo deve aprender com esses dados e ser capaz de prever o preço médio em qualquer bairro, considerando todas as outras métricas.*

Descrição da Tarefa

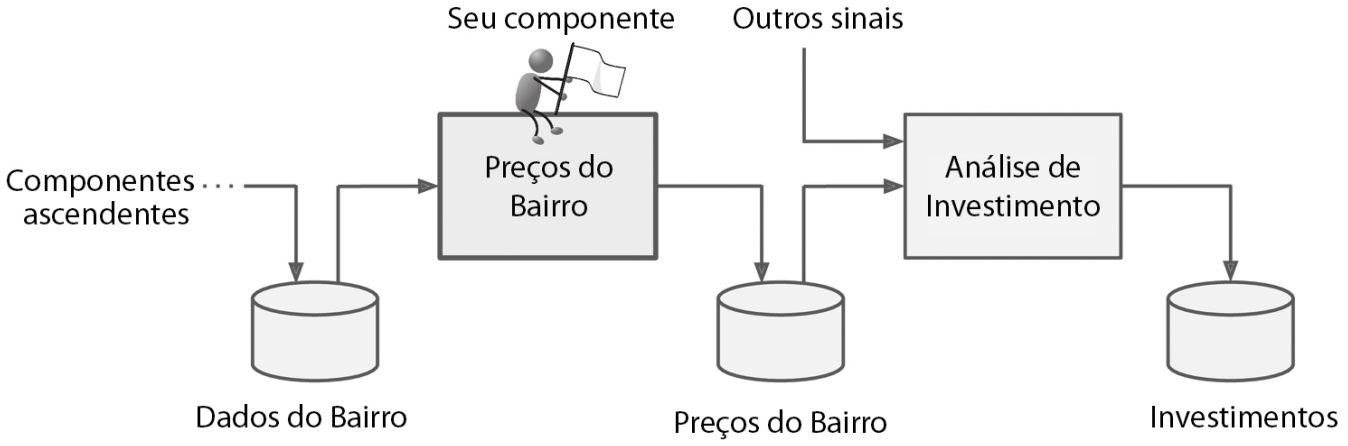


Figura - pipeline de Aprendizado de Máquina

Pipeline de dados: sequência de componentes de processamento de dados

- Componentes rodam de forma assíncrona;
- O armazenamento de dados é a interface entre os componentes;
- Cada componente é autônomo:
    - Torna a arquitetura robusta;
    - Simplifica o entendimento;
    - Permite a divisão do trabalho;

- Requer monitoramento adequado (um componente rompido causará queda no desempenho, devido a dados obsoletos),

Solução Atual: estimativa manual realizada por especialistas. Processo caro, demorado e com estimativas não boas (10% abaixo do valor).

Enquadramento do Problema:

- **Quanto à Supervisão**: Treinamento Supervisionado (dados rotulados)
- **Quanto à Tarefa**: Regressão multivariada (previsão de valor contínuo com múltiplas características)
- **Quanto ao Modo**: Aprendizado em Lote (não há fluxo contínuo de dados, não há necessidade de ajuste rápido a mudança de dados, dados são pequenos para memória)

Selecione uma Medida de Desempenho: Raiz do Erro Quadrático Médio (RMSE)

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

- m - número de instâncias do conjunto de dados.
- xi - vetor de todos os valores da característica (excluindo o rótulo).
- yi - rótulo (valor desejado da saída).
- X - matriz que contém todos os valores da característica (excluindo rótulos) de todas as instâncias do conjunto de dados.
- h (hipótese) - função de previsão do sistema (valor previsto).
- RMSE (X,h) - função de custo medida no conjunto de exemplos X, utilizando a hipótese h.

B) Configuração

Primeiro, vamos importar alguns módulos comuns, garantir que o MatplotLib plote as figuras inline e prepare uma função para salvar as figuras. Também verificamos se o Python 3.5 ou posterior está instalado (embora o Python 2.x possa funcionar, ele está obsoleto, portanto, recomendamos fortemente que você use o Python 3), bem como o Scikit-Learn ≥0.20.

```
# Python ≥3.5 é obrigatório
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 é obrigatório
import sklearn
assert sklearn.__version__ >= "0.20"

```

```
# Importações comuns
import numpy as np
import os
```

```
# Para traçar figuras bonitas
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

```
# Onde salvar as figuras
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

ETAPA 02 - Análise dos Dados

A) Obtenha os Dados

Baixe os Dados

```
import os
import tarfile
import urllib.request

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

# Função para criar um diretório datasets/housing no seu espaço de trabalho, baixa o arquivo h
def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")

    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

```
fetch_housing_data()
```

```
# Carregar os dados com Pandas
import pandas as pd

# Função que retorna um objeto DataFrame Pandas contendo todos os dados
def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

Visualize a Estrutura de Dados

Cada linha representa um distrito. "total_rooms" é o total de cômodos do distrito. "households" é o total de residências.

```
housing = load_housing_data()
# Para ver as 5 linhas superiores
housing.head()
```

Cada linha representa um bairro. Existem 10 atributos.

```
# Para obter uma rápida descrição dos dados (Nr total de linhas, tipo de cada atributo e o nr
housing.info()
```

Observações:

- O atributo total_bedrooms possui 207 valores nulos
- O tipo do atributo ocean_proximity é object (categorico)

```
housing["ocean_proximity"].value_counts()
```

```
# Descreve os atributos numéricos (medidas estatísticas)
housing.describe()
```

```
# Constroi um histograma para cada atributo numérico
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

Observações:

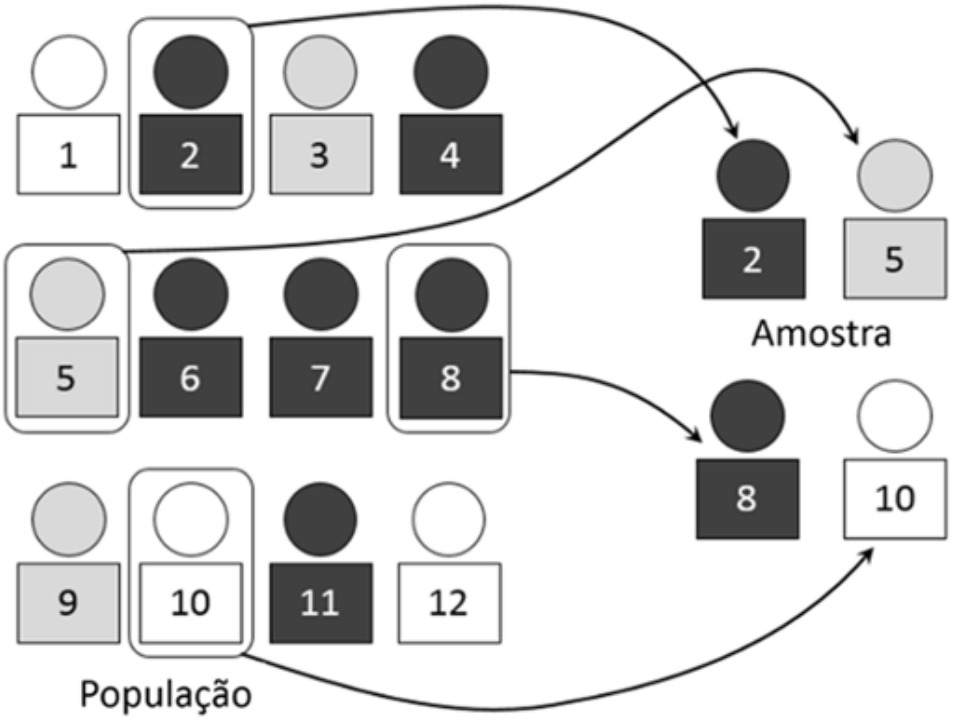
- **Problema 1** - Atributo housing_median_age e median_house_value (atributo alvo) foi limitado e isso poderá ser um problema caso o sistema tenha de prever valores superiores, pois o algoritmo poderá entender que o valor nunca ultrapassará este limite.
    - Duas opções:
        - Coletar novos dados com valores superiores

        - Remover os dados limitados do conjunto de dados.
- **Problema 2**: Os atributos tem escalas muito diferentes, o que irá requerer o Escalonamento de Características.
- **Problema 3**: muitos histogramas tem um rastro alongado - eles se estendem muito mais à direita da média do que à esquerda, o que vai requerer uma Transformação de Atributo.

B) Crie um Conjunto de Teste

Amostragem Aleatória Simples

- Seleção aleatória de indivíduos do Universo com idêntica probabilidade de pertencer à Amostra



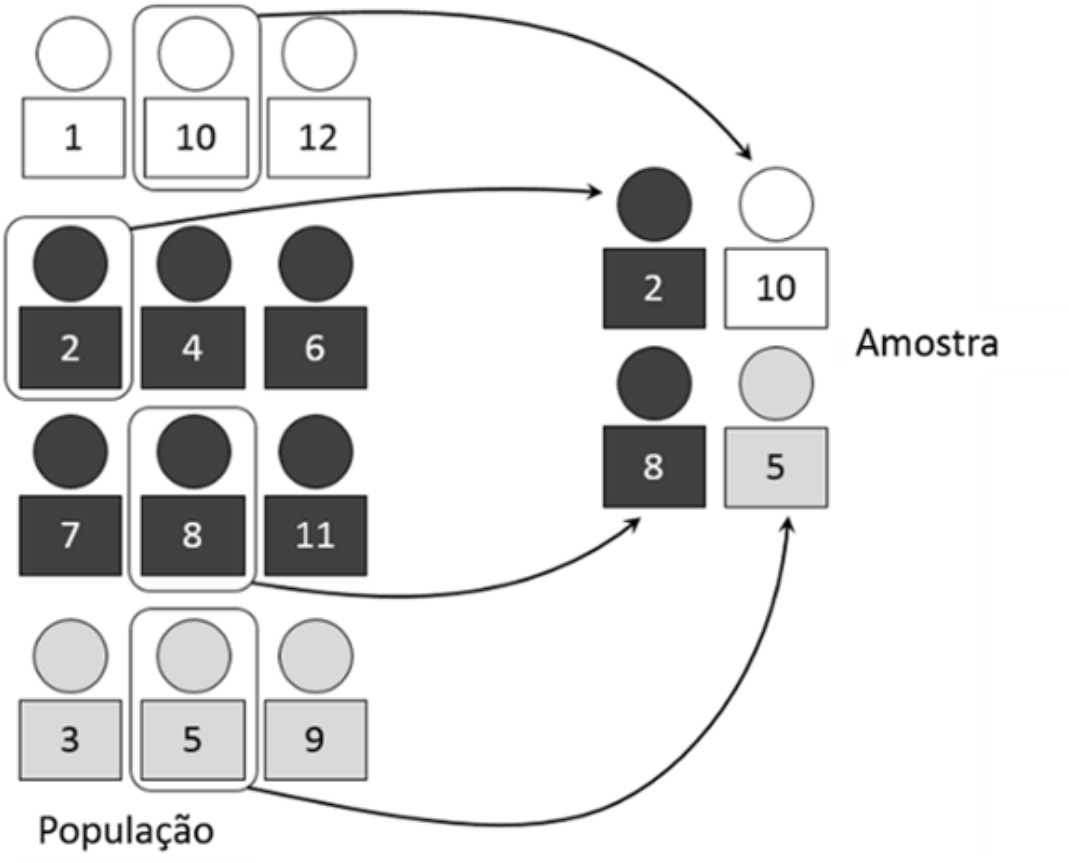
Usando Funções do Scikit-Learn para dividir o conjunto de dados com amostragem aleatória

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42) # define o a s
```

```
test_set.head()
```

Amostragem Estratificada

- Seleção de indivíduos de Grupos internamente homogêneos com qualquer técnica de



Amostragem.

Considerando que a renda média (income_cat) é um atributo muito importante para estimar os preços médios. É necessário garantir que o conjunto de testes seja representativo das várias categorias de rendimentos em todo o conjunto de dados.

```
housing["median_income"].hist()
```

```
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])
```

```
housing["income_cat"].value_counts()
```

```
housing["income_cat"].hist()
```

Usando Funções do Scikit-Learn para dividir o conjunto de dados com amostragem estratificada baseado no atributo renda média (income_cat)

```
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]
```

Comparando as proporções da categoria de renda no conjunto gerado com a amostragem estratificada e no conjunto completo de dados.

```
# proporções da categoria de renda no conjunto gerado com a amostragem estratificada
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

```
# proporções da categoria de renda no conjunto completo de dados
housing["income_cat"].value_counts() / len(housing)
```

```
# Criando a tabela de comparação de Viés de Amostragem
def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)

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

# tabela de comparação de Viés de Amostragem
compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_test_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_props["Random"] / compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] / compare_props["Overall"]
```

```
# Comparação de viés de amostragem estratificada versus amostragem aleatória
compare_props
```

```
# Removendo o atributo income_cat para que os dados voltem ao seu estado original
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

C) Visualize os Dados para Obter Informações

```
# Criando uma cópia do conjunto de treinamento para manipulação segura
housing = strat_train_set.copy()
```

Visualizando Dados Geográficos

```
# Criando um diagrama de dispersão para visualizar os dados de todos os bairros (informações g
housing.plot(kind="scatter", x="longitude", y="latitude")
```

```
# A opção Alpha em 0.1 facilita a visualização dos locais onde existe uma alta densidade de po
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

Vejamos os preços do setor imobiliário. O raio de cada círculo representa a população do bairro (opção s) e a cor representa o preço (opção c). Usaremos um mapa de cores prédefinido (opção cmap) chamado jet, que varia do azul (valores baixos) para vermelho (preços altos).

```
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,
             sharex=False)
# O argumento sharex=False corrige um erro de exibição (os valores e a legenda do eixo x não f
# Esta é uma correção temporária (consulte: https://github.com/pandas-dev/pandas/issues/10611
plt.legend()
```

Esta imagem informa que os preços do setor imobiliário estão muito relacionados à localização e a densidade populacional. Será útil utilizar um algoritmo de agrupamento para detectar os grupos principais e adicionar novas características que medem a proximidade com os centros de agrupamento.

```
# Para baixar a imagem da Califórnia
images_path = os.path.join(PROJECT_ROOT_DIR, "images", "end_to_end_project")
os.makedirs(images_path, exist_ok=True)

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
filename = "california.png"
print("Downloading", filename)

url = DOWNLOAD_ROOT + "images/end_to_end_project/" + filename
# Copia um objeto de rede indicado por uma URL para um arquivo local
urllib.request.urlretrieve(url, os.path.join(images_path, filename))
```

```
import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                  s=housing['population']/100, label="Population",
                  c="median_house_value", cmap=plt.get_cmap("jet"),
                  colorbar=False, alpha=0.4)
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)
# Preparando a Legenda
prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cbar = plt.colorbar(ticks=tick_values/prices.max())
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
plt.show()
```

Buscando Correlações

```
# Calculando o Coeficiente de Correlação Padrão (r de Pearson)
corr_matrix = housing.corr()
```

O coeficiente de correlação varia de -1 a 1. Próximo de 1 é uma forte correlação positiva, próximo de -1 é uma forte correlação negativa e próximo de zero significa que não há correlação linear. Observe que o

valor médio da habitação (median_house_value) tende a aumentar quando a renda média (median_income) aumenta e existe uma pequena correlação negativa entre a latitude e o valor médio da habitação (os preços tendem a diminuir em direção ao norte).

```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Outra maneira de verificar a correlação entre atributos é utilizar a função scatter_matrix, do Pandas, que plota cada atributo numérico em relação a qualquer outro atributo numérico (11 atributos = 121 plotagens). Então, selecionamos apenas alguns atributos promissores que parecem mais correlacionados com o valor médio do setor imobiliário.

```
# from pandas.tools.plotting import scatter_matrix # For older versions of Pandas
from pandas.plotting import scatter_matrix

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

O atributo mais promissor para prever o valor médio da habitação (median_house_value) é a renda média (median_income) , então vamos observar o gráfico de dispersão de correlação.

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
```

A plotagem revela uma forte correlação com tendência ascendente e o a presença do limite artificial de preços visível como uma linha horizontal em 500.000

Experimentando Combinações de Atributos

Criando novos atributos:

- Número de cômodos por domicílio (rooms_per_household)
- Número de quartos por cômodos (bedrooms_per_room)
- População por domicílio (population_per_household)

```
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"]
```

```
# Calculando o Coeficiente de Correlação Padrão (r de Pearson)
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

O novo atributo número de quartos por cômodos (bedrooms_per_room) está muito mais correlacionado com o valor médio da habitação (median_house_value) do que o número total de cômodos (total_rooms)

ou número total de quartos (total_bedrooms).

```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```

```
housing.describe()
```

D) Prepare os Dados para Algoritmos do Aprendizado de Máquina

Revertendo para um conjunto de treinamento limpo

```
# Separando os atributos previsores e os rótulos, removendo os rótulos
housing = strat_train_set.drop("median_house_value", axis=1)
# Criando uma cópia dos rótulos
housing_labels = strat_train_set["median_house_value"].copy()
```

Limpando os Dados

O atributo total _bedrooms possui valores ausentes, existem 3 opções para solucionar o problema:

- **opção 1** - eliminar os bairros correspondentes (linhas);
  - housing.dropna(subset=["total_bedrooms"])
- **opção 2** - eliminar o atributo (coluna)
  - housing.drop("total_bedrooms", axis=1)
- **opção 3** - atribuir um valor para os valores ausentes.
  - median = housing["total_bedrooms"].median()
  - housing["total_bedrooms"].fillna(median, inplace=True)

Para demonstrar cada um deles, vamos criar uma cópia do conjunto de dados de alojamento, mas mantendo apenas as linhas que contêm pelo menos um nulo. Assim será mais fácil visualizar exatamente o que cada opção faz:

```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # opção 1
```

```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # opção 2
```

```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # opção 3
sample_incomplete_rows
```

O Scikit-Learn fornece uma classe acessível para cuidar dos valores faltantes: Imputer

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

Remova o atributo de texto porque a mediana só pode ser calculada em atributos numéricos:

```
housing_num = housing.drop("ocean_proximity", axis=1)
# alternatively: housing_num = housing.select_dtypes(include=[np.number])
```

```
# ajustando a instância imputer aos dados de treinamento utilizando o método fit()
imputer.fit(housing_num)
```

```
# Visualizando o método de transformação empregado
imputer.strategy
```

```
# Imputer armazena o resultado em sua variável de instância statistics_
imputer.statistics_
```

Verifique se isso é o mesmo que calcular manualmente a mediana de cada atributo:

```
housing_num.median().values
```

Agora, você pode utilizar este imputer treinado substituindo os valores ausentes pelas médias aprendidas, afim de transformar o conjunto de treinamento:

```
X = imputer.transform(housing_num)
```

```
# Inserindo o resultado em um DataFrame Pandas
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

```
# Visualizando os valores das médias inseridos no atributo total_bedrooms
housing_tr.loc[sample_incomplete_rows.index.values]
```

```
housing_tr.info()
```

Manipulando Texto e Atributos Categóricos

Agora vamos pré-processar o recurso de entrada categórica, ocean_proximity, convertendo essas categorias de texto para números:

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

Convertendo o rótulo das categorias em numeros inteiros

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

```
ordinal_encoder.categories_
```

Um problema nesta representação é que os algoritmos de AM assumirão que dois valores próximos são mais parecidos do que dois valores distantes (por exemplo, as categorias 0 e 2 serem mais parecidas do que 0 e 4, o que não é verdadeiro nesta base de dados)

O Scikit-Learn fornece um codificador OneHotEncoder para converter valores categóricos inteiros em vetores one-hot (um atributo será igual a 1[hot], enquanto os outros serão 0 [cold]

Exemplo

Original Data

Team	Points
A	25
A	12
B	15
B	14
B	19
B	23
C	25
C	29

One-Hot Encoded Data

Team_A	Team_B	Team_C	Points
1	0	0	25
1	0	0	12
0	1	0	15
0	1	0	14
0	1	0	19
0	1	0	23
0	0	1	25
0	0	1	29

```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Por padrão, a classe OneHotEncoder retorna um array esparso (matriz que armazena apenas a localização dos elementos diferentes de zero), mas podemos convertê-lo em um array denso, se necessário, chamando o método toarray():

```
housing_cat_1hot.toarray()
```

Alternativamente, você pode definir sparse=False ao criar o OneHotEncoder:

```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

```
# Obtendo a lista de categorias
cat_encoder.categories_
```

Transformadores Personalizados

Embora o Scikit-Learn forneça muitos transformadores úteis, você precisará escrever seus próprios para tarefas como operações de limpeza personalizadas ou combinar atributos específicos. Você vai querer que seu transformador funcione perfeitamente com as funcionalidades do Scikit-Learn (como pipelines).

Tudo que você precisa fazer é criar uma classe e implementar três métodos : fit() (retornando self), transform() e fit_transform().

Você pode obter o último gratuitamente simplesmente adicionando TransformerMixin como uma classe base.

Se você adicionar BaseEstimator como uma classe base (e evitar *args e *kargs em seu construtor), você também obterá dois métodos extras (get_params() e set_params()) que serão úteis para ajuste automático de hiperparâmetros.

Vamos criar um transformador personalizado para adicionar atributos extras (bedrooms_per_room, rooms_per_household, population_per_household):

```
from sklearn.base import BaseEstimator, TransformerMixin
# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # sem *args ou **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nada mais a fazer
    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,
                         bedrooms_per_room]
        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)
```

Observe que codifiquei os índices (3, 4, 5, 6) para concisão e clareza, mas seria muito mais fácil obtê-los dinamicamente, assim:

```
col_names = "total_rooms", "total_bedrooms", "population", "households"
rooms_ix, bedrooms_ix, population_ix, households_ix = [
    housing.columns.get_loc(c) for c in col_names] # pega o indice das colunas
```

Além disso, housing_extra_attribs é um array NumPy, perdemos os nomes das colunas (infelizmente, isso é um problema com o Scikit-Learn). Para recuperar um DataFrame, você pode executar isto:

```
housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household", "population_per_household"],
    index=housing.index)
housing_extra_attribs.head()
```

Escalonamento de Características

Uma das transformações mais importantes que você precisa aplicar aos seus dados é o redimensionamento de características.

Com poucas exceções, os algoritmos de Machine Learning não funcionam bem quando os atributos numéricos de entrada têm escalas muito diferentes.

Este é o caso para os dados de habitação: o número total de quartos varia de cerca de 6 a 39.320, enquanto os rendimentos médios variam apenas de 0 a 15. Observe que a escala dos valores-alvo geralmente não é necessária.

Escala min-max (Normalização)

Define uma nova escala de valores, limites mínimos e máximos, para todos os atributos.

$$V_{novo} = min + \frac{V_{atual} - menor}{maior - menor}(max - min)$$

Exemplo do MinMaxScaler (normalização)

```
from sklearn.preprocessing import MinMaxScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
```

```
scaler = MinMaxScaler()
scaler.fit(data)
```

```
print(scaler.data_max_)
```

```
print(scaler.data_min_)
```

```
print(scaler.transform(data))
```

Exemplo do StandardScaler (padronização)

Padronização (escore-z)

A Cada valor do atributo a ser normalizado é adicionada ou subtraída uma medida de localização (valor médio) e o valor resultante é multiplicado ou dividido por uma medida de escala (desvio padrão).

A padronização subtrai o valor médio (portanto, os valores padronizados sempre têm média zero) e depois divide pelo desvio padrão para que a distribuição resultante tenha variância unitária.

$$V_{novo} = \frac{V_{atual} - \mu}{\sigma}$$

Ao contrário do escalonamento min-max, a padronização não limita valores a um intervalo específico, o que pode ser um problema para alguns algoritmos (por exemplo, redes neurais geralmente esperam um valor de entrada variando de 0 a 1). No entanto, a padronização é muito menos afetada por outliers.

```
from sklearn.preprocessing import StandardScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = StandardScaler()
print(scaler.fit(data))
```

```
print(scaler.transform(data))
```

Pipelines de Transformação

Agora vamos construir um pipeline para pré-processar os atributos numéricos:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
# valores ausentes, combinando atributos, escalonamento
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
housing_num_tr
```

Até agora, tratamos as colunas categóricas e as colunas numéricas separadamente.

Seria mais conveniente ter um único transformador capaz de lidar com todas as colunas, aplicando as transformações apropriadas a cada coluna.

O Scikit-Learn tem o **ColumnTransformer** para tratar colunas numéricas e categóricas. Vamos usá-lo para aplicar todas as transformações aos dados da habitação:

```
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)
```

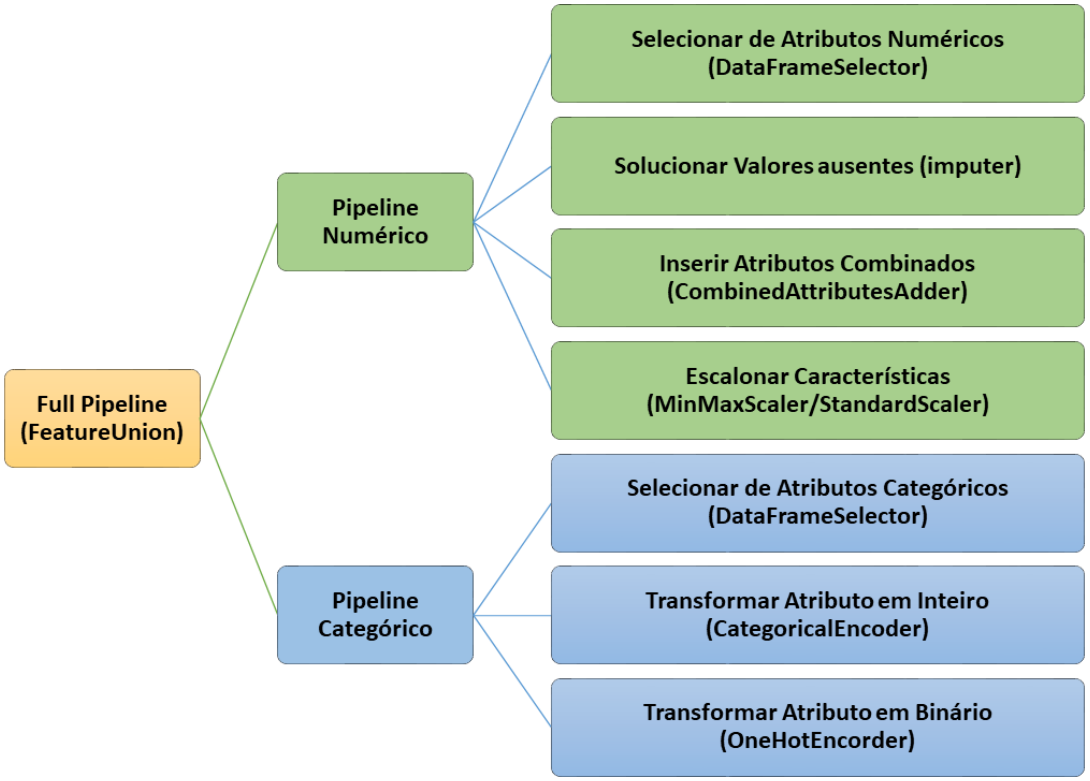
```
housing_prepared
```

```
housing_prepared.shape
```

Para referência, aqui está a solução antiga baseada em um transformador DataFrameSelector (para apenas selecionar um subconjunto das colunas Pandas DataFrame) e um FeatureUnion:

```
from sklearn.base import BaseEstimator, TransformerMixin
# Cria uma classe para selecionar colunas numéricas ou categóricas
class OldDataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

Agora vamos juntar todos esses componentes em um grande pipeline que pré-processará os recursos numéricos e categóricos:




```
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]
# pipeline para atributos numéricos
old_num_pipeline = Pipeline([
        ('selector', OldDataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])
# pipeline para atributos categóricos
old_cat_pipeline = Pipeline([
        ('selector', OldDataFrameSelector(cat_attribs)),
        ('cat_encoder', OneHotEncoder(sparse=False)),
    ])
```

```
# Juntando os dois pipeline em um único pipeline
from sklearn.pipeline import FeatureUnion

old_full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", old_num_pipeline),
        ("cat_pipeline", old_cat_pipeline),
    ])
```

```
old_housing_prepared = old_full_pipeline.fit_transform(housing)
old_housing_prepared
```

```
np.allclose(housing_prepared, old_housing_prepared)
```

ETAPA 03 - Construção do Modelo

A) Selecione e Treine um Modelo

Treinando e Avaliando o Conjunto de Treinamento com o modelo de Regressão Linear

```
# Treinando o modelo de Regressão Linear
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

```
# vamos experimentar o pipeline de pré-processamento completo em algumas instâncias de treinam
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))
```

Compare com os valores reais:

```
print("Labels:", list(some_labels))
```

```
some_data_prepared
```

```
# Avaliando o modelo de Regressão Linear com a raiz do erro médio quadrático
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
```

Isso é melhor do que nada, mas claramente não é uma ótima pontuação: os valores médios da maioria dos distritos variam entre US 120.000 e US$ 265.000, portanto, um erro de previsão típico de US 68.628 não é muito satisfatório. Este é um exemplo de um modelo que não ajusta os dados de treinamento.

Quando isso acontece, pode significar que os recursos não fornecem informações suficientes para fazer boas previsões ou que o modelo não é poderoso o suficiente.

Nota: *desde o Scikit-Learn 0.22, você pode obter o RMSE diretamente chamando a função mean_squared_error() com squared=False.*

```
# avaliando o erro absoluto médio (MAE)
from sklearn.metrics import mean_absolute_error
lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

Treinando e Avaliando o Conjunto de Treinamento com o modelo de Ávores de Decisão

Vamos treinar um **DecisionTreeRegressor**. Este é um modelo poderoso, capaz de encontrar relacionamentos não lineares complexos nos dados (**Árvores de Decisão** serão apresentadas com mais detalhes mais a frente).

```
# Treinando o modelo de Árvore de Decisão
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

```
# avaliando o modelo de Árvore de Decisão com a raiz do erro médio quadrático
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Observação: Nenhum erro? Poderia este modelo realmente ser absolutamente perfeito? Claro, é muito mais provável que o modelo tenha se sobreajustado aos dados.

Como você pode ter certeza? Como vimos anteriormente, você não deseja tocar no conjunto de teste até que esteja pronto para lançar um modelo no qual esteja confiante, portanto, você precisa usar parte do conjunto de treinamento para treinamento e parte dele para validação do modelo.

Avaliando Melhor com a Utilização da Validação Cruzada

Uma maneira de avaliar o modelo de árvore de decisão seria usar a função **train_test_split()** para dividir o conjunto de treinamento em um **conjunto de treinamento menor** e um **conjunto de validação**.

Em, seguida, treinar seus modelos em relação ao conjunto de treinamento menor e avaliá-los em relação ao conjunto de validação. É um pouco trabalhoso, mas nada muito difícil, e funcionaria muito bem.

Uma ótima alternativa é usar o recurso de **validação cruzada K-fold** do Scikit-Learn.

O código a seguir divide aleatoriamente o conjunto de treinamento em 10 subconjuntos distintos chamados pastas, depois treina e avalia o modelo de Árvore de Decisão 10 vezes, escolhendo uma pasta diferente para avaliação a cada vez e treinando nas outras 9 pastas.

O resultado é um array contendo as 10 pontuações de avaliação:

```
# Treinando e Avaliando o modelo Árvore de Decisão com Validação Cruzada
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) # A função de pontuação é o oposto do MSE (-scores)
```

```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

Conclusão: Agora, a Árvore de Decisão não parece tão boa quanto antes. Na verdade, parece ter um desempenho pior do que o modelo de Regressão Linear!

Observe que a validação cruzada permite que você obtenha não apenas uma estimativa do desempenho do seu modelo, mas também uma medida de quão precisa é essa estimativa (ou seja, seu desvio padrão).

Calcularemos as mesmas pontuações para o modelo de Regressão Linear apenas para ter certeza.

```
# Treinando e Avaliando o modelo de Regressão Linear com Validação Cruzada
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) # A função de pontuação é o oposto do MSE (-scores)
display_scores(lin_rmse_scores)
```

```
# Descrevendo detalhes da Validação Cruzada para o modelo de Regressão Linear
scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

Conclusão: o modelo de Árvore de Decisão está se ajustando tanto que tem um desempenho pior do que o modelo de Regressão Linear.

Treinando e Avaliando o Conjunto de Treinamento com o modelo de Florestas Aleatórias

As Florestas Aleatórias funcionam treinando muitas Árvores de Decisão em subconjuntos aleatórios dos recursos e, em seguida, calculando a média de suas previsões.

Construir um modelo em cima de muitos outros modelos é chamado de **Ensemble Learning** e geralmente é uma ótima maneira de impulsionar ainda mais os algoritmos de ML.

```
# Treinando o modelo de Florestas Aleatórias
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

Nota: *especificamos n_estimators=100 para ser à prova de futuro, pois o valor padrão mudará para 100 no Scikit-Learn 0.22.*

```
# Avaliando o modelo de Florestas Aleatórias com a raiz do erro médio quadrático
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
# Avaliando o modelo de Florestas Aleatórias com Validação Cruzada
from sklearn.model_selection import cross_val_score
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

Conclusão: Florestas Aleatórias parecem muito promissoras. No entanto, observe que a pontuação no conjunto de treinamento ainda é muito menor do que nos conjuntos de validação, o que significa que o modelo ainda está superajustando o conjunto de treinamento. As soluções possíveis para o overfitting são simplificar o modelo, restringi-lo (ou seja, regularizá-lo) ou obter muito mais dados de treinamento.

Antes de mergulhar muito mais fundo nas Florestas Aleatórias, no entanto, você deve experimentar muitos outros modelos de várias categorias de algoritmos de Aprendizado de Máquina (por exemplo, várias máquinas de vetor de suporte com kernels diferentes e possivelmente uma rede neural), sem gastar muito tempo ajustando os hiperparâmetros. O objetivo é selecionar alguns (dois a cinco) modelos promissores.

Treinando e Avaliando o Conjunto de Treinamento com o modelo SVR

```
# Treinando o modelo de SVR e Avaliando com a raiz do erro médio quadrático
from sklearn.svm import SVR
svm_reg = SVR(kernel="linear")
svm_reg.fit(housing_prepared, housing_labels)
housing_predictions = svm_reg.predict(housing_prepared)
svm_mse = mean_squared_error(housing_labels, housing_predictions)
svm_rmse = np.sqrt(svm_mse)
svm_rmse
```

ETAPA 04 - Avaliação do Modelo

A) Ajuste seu Modelo

Vamos supor que agora você tenha uma lista de modelos promissores. Agora você precisa ajustá-los. Vejamos algumas maneiras de fazer isso.

Pesquisa de grade (Grid Search) - Abordagem para poucas combinações de Hiperparâmetros

Uma opção seria mexer nos hiperparâmetros manualmente, até encontrar uma ótima combinação de valores de hiperparâmetros. Isso seria um trabalho muito tedioso e você pode não ter tempo para explorar muitas combinações.

Em vez disso, você deve obter o **GridSearchCV** do Scikit-Learn para pesquisar por você.

Tudo o que você precisa fazer é dizer quais hiperparâmetros você deseja experimentar e quais valores experimentar, e ele usará validação cruzada para avaliar todas as combinações possíveis de valores de hiperparâmetros.

Por exemplo, o código a seguir procura a melhor combinação de valores de hiperparâmetros para RandomForestRegressor:

```
# Pesquisando a melhor combinação de valores de hiperparâmetros
from sklearn.model_selection import GridSearchCV
param_grid = [
    # tente 12 (3 × 4) combinações de hiperparâmetros
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # em seguida, tente 6 (2 × 3) combinações com bootstrap definido como False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# treine em 5 pastas, que é um total de (12+6)*5=90 rodadas de treinamento
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)
```

A pesquisa em grade explorará 12 + 6 = 18 combinações de valores de hiperparâmetros RandomForestRegressor e treinará cada modelo 5 vezes (já que estamos usando validação cruzada de cinco vezes). Em outras palavras, ao todo, serão 18 × 5 = 90 rodadas de treinamento! Pode levar muito tempo, mas quando estiver pronto, você poderá obter a melhor combinação de parâmetros como este:

```
# A melhor combinação de hiperparâmetros encontrada
grid_search.best_params_
```

Como 8 e 30 são os valores máximos avaliados, você provavelmente deve tentar pesquisar novamente com valores mais altos; a pontuação pode continuar a melhorar.

Alem disso, voce tem acesso ao melhor estimador (modelo) diretamente

```
# melhores hiperparâmetros encontrados para o estimador
```

```
grid_search.best_estimator_
```

Vejamos a pontuação de cada combinação de hiperparâmetros testada durante a pesquisa de grade:

```
# Pontuação de cada combinação de hiperparâmetros testada com grid_search
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

Melhor Resultado: 49898.98913455217 {'max_features': 8, 'n_estimators': 30}

```
# Resultado detalhado das pontuações com grid_search
pd.DataFrame(grid_search.cv_results_)
```

Neste exemplo, obtemos a melhor solução definindo o hiperparâmetro **max_features** como **8** e o hiperparâmetro **n_estimators** como **30**.

A pontuação RMSE para essa combinação é um pouco melhor do que a pontuação obtida anteriormente usando os **valores padrão do hiperparâmetro** .

Comparação dos métodos (modelo RandomForestRegressor):

Melhor Resultado (Valores Padrão): 50435.58092066179

Melhor Resultado (Grid_Search): 49898.98913455217 {'max_features': 8, 'n_estimators': 30}

Observação: Não se esqueça de que **você pode tratar algumas das etapas de preparação de dados como hiperparâmetros**. Por exemplo, a pesquisa em grade descobrirá automaticamente se deve ou não adicionar um recurso sobre o qual você não tem certeza (por exemplo, usando o hiperparâmetro add_bedrooms_per_room do seu transformador CombinedAttributesAdder). Da mesma forma, pode ser usado para encontrar automaticamente a melhor maneira de lidar com valores discrepantes, recursos ausentes, seleção de recursos e muito mais.

Pesquisa Aleatória (Randomized Search) - Abordagem para muitas combinações de Hiperparâmetros

Quando o espaço de busca do Hiperparâmetro for grande é preferível utilizar o **Randomizer Search** em vez de **Grid Search**. Em vez de tentar todas as combinações possíveis, esta classe escolhe um valor aleatório para cada hiperparâmetro em cada interação e avalia um determinado número de combinações aleatórias.

```
# Pesquisando a melhor combinação de valores de hiperparâmetros
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

```

```
param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', random_stat
rnd_search.fit(housing_prepared, housing_labels)
```

```
# Pontuação de cada combinação de hiperparâmetros testada com RandomizedSearch
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

Melhor Resultado: 49117.55344336652 {'max_features': 7, 'n_estimators': 180}

Comparação dos métodos (modelo RandomForestRegressor):

Melhor Resultado (Valores Padrão): 50435.58092066179

Melhor Resultado (Grid_Search): 49898.98913455217 {'max_features': 8, 'n_estimators': 30}

Melhor Resultado (RandomizedSearch): 49117.55344336652 {'max_features': 7, 'n_estimators': 180}

Analise os Melhores Modelos e seus Erros

Muitas vezes você obterá boas ideias sobre o problema ao inspecionar os melhores modelos. Por exemplo, random Forest regressor pode indicar a importância relativa de cada tributos para fazer previsões precisas:

```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Mostraremos essas pontuações de importância ao lado de seus nomes de atributos correspondentes:

```
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
#cat_encoder = cat_pipeline.named_steps["cat_encoder"] # solução antiga
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)
```

Com esta informação, você pode descartar algumas das características menos úteis (por exemplo, aparentemente apenas uma categoria ocean_proximity ['INLAND'] é realmente útil, você pode tentar descartar as outras).

Você também deve analisar os erros específicos cometidos pelo seu sistema, depois tentar entender porque eles os faz e o que poderia solucionar o problema (adicionar características extras, se livrar de

características não informativas, limpar outliers, etc).

Avalie seu Sistema no Conjunto de Testes

Depois de ajustar seus modelos agora é a hora de avaliar o modelo final no conjunto de teste. Não há nada de especial neste processo; apenas obtenha as previsões e os rótulos do seu conjunto de teste, execute full_pipeline para transformar os dados (chame transform(), não fit_transform()) e avalie o modelo final no conjunto de teste.

```
final_model = grid_search.best_estimator_
# Separação dos atributos de entrada e de saída
X_test = strat_test_set.drop("median_house_value", axis=1) # dados sem rótulos
y_test = strat_test_set["median_house_value"].copy() # rótulos

X_test_prepared = full_pipeline.transform(X_test) # preparação dos dados
final_predictions = final_model.predict(X_test_prepared) # predição
# avaliação com RMSE
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
final_rmse
```

O desempenho geralmente será um pouco pior do que o medido usando validação cruzada se você fez muitos ajustes de hiperparâmetros porque o sistema acabou sendo ajustado para executar bem com dados de validação e provavelmente não funcionará tão bem em conjuntos desconhecidos de dados. Não é o caso neste exemplo, mas, quando isso acontece, você não deve ajustar os hiperparâmetros para que os números fiquem mais atraentes no conjunto de teste; as melhorias não seriam generalizadas para novos dados.

Outros medidas:

Podemos calcular um intervalo de confiança de 95% para o teste RMSE:

```
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)))
```

Poderíamos calcular o intervalo manualmente assim:

```
m = len(squared_errors)
mean = squared_errors.mean()
tscore = stats.t.ppf((1 + confidence) / 2, df=m - 1)
```

```
tmargin = tscore * squared_errors.std(ddof=1) / np.sqrt(m)
```

Alternativamente, poderíamos usar um z-scores em vez de t-scores:

```
zscore = stats.norm.ppf((1 + confidence) / 2)
zmargin = zscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - zmargin), np.sqrt(mean + zmargin)
```

ETAPA 05 - Implantação e Manutenção do Sistema

A) Apresentação da Solução

Agora, vem a fase de pré-lançamento do projeto. você precisa apresentar sua solução (destacando o que aprendeu, o que funcionou e o que não funcionou, quais pressupostos foram feitos e quais as limitações do seu sistema); documente tudo e crie apresentações detalhadas com visualizações claras e declarações que sejam fáceis de lembrar (por exemplo, "a renda média é o principal previsor dos preços do setor imobiliário").

B) Lançamento, Monitoração e Manutenção do Sistema

Após aprovação para o lançamento você precisa preparar sua solução para produção, principalmente conectando as fontes de dados de entrada da produção ao seu sistema e escrevendo testes.

Você também precisa escrever o código de monitoramento para verificar o desempenho do sistema em tempo real em intervalos regulares e adicionar alertas quando ele ficar offline. Isso é importante, não apenas para capturar uma queda súbita, mas também a degradação do desempenho. Isso é bastante comum porque os modelos tendem a deteriorar a medida que os dados evoluem ao longo do tempo, a menos que sejam regularmente treinados em novos dados.

Avaliação do desempenho do seu sistema exigirá uma amostragem das previsões do sistema e sua avaliação. Isto geralmente requer uma análise humana. Esses analistas podem ser especialistas, de qualquer forma, você precisará conectar o canal de avaliação humana ao seu sistema.

Você também deve certificar-se de avaliar a qualidade de dados de entrada do sistema. Às vezes, se deteriorar levemente sinal de má qualidade, demorar até que o desempenho se de grade o suficiente para disparar um alerta. O monitoramento das entradas é particularmente importante para os sistemas de aprendizado online.

Finalmente, treinar regularmente seus modelos com utilização de novos dados. Esse processo deve ser automatizado tanto quanto possível.

Exercício

Experimente a metodologia apresentada para preparar os dados para previsao de alugueis do dataset em https://www.kaggle.com/datasets/rubenssjr/brasilian-houses-to-rent?select=houses_to_rent_v2.csv