



EQUIPE 1

Desafio-IA Machine Learning

Membros

Vitor Nogueira de Sousa - Ciência da Computação

Artur Carrah Cerqueira - Engenharia da Computação

Victor de Oliveira Bezerra- Ciência da Computação

Mateus Andrade Maia - Engenharia da Computação



EQUIPE 1

Desafio

Será proposto um desafio de implementação de um algoritmo voltado para a previsão de valores a partir de um banco de dados. O objetivo é utilizar um modelo de aprendizado de máquina capaz de aprender padrões históricos nos dados e, após um processo de treinamento, realizar previsões com base em novos conjuntos de dados.

- Banco de Dados(Dataset)
- Calirfonia Housing Prices
- Kaggle
- Python for Data Science
- Intro to Machine Learning
- Pandas
- Matplotlib
- Sklearn
- Colab
- MAE and MAPE
- Decision Tree
- Random Forest



EQUIPE 1

Dataset

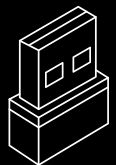
California Housing Prices

Os dados dizem respeito às casas encontradas em um determinado distrito da Califórnia e algumas estatísticas resumidas sobre elas com base nos dados do censo de 1990. Embora eles possam não ajudar a prever os preços atuais de moradias, eles fornecem um conjunto de dados introdutório acessível para ensinar as pessoas sobre os fundamentos do aprendizado de máquina.



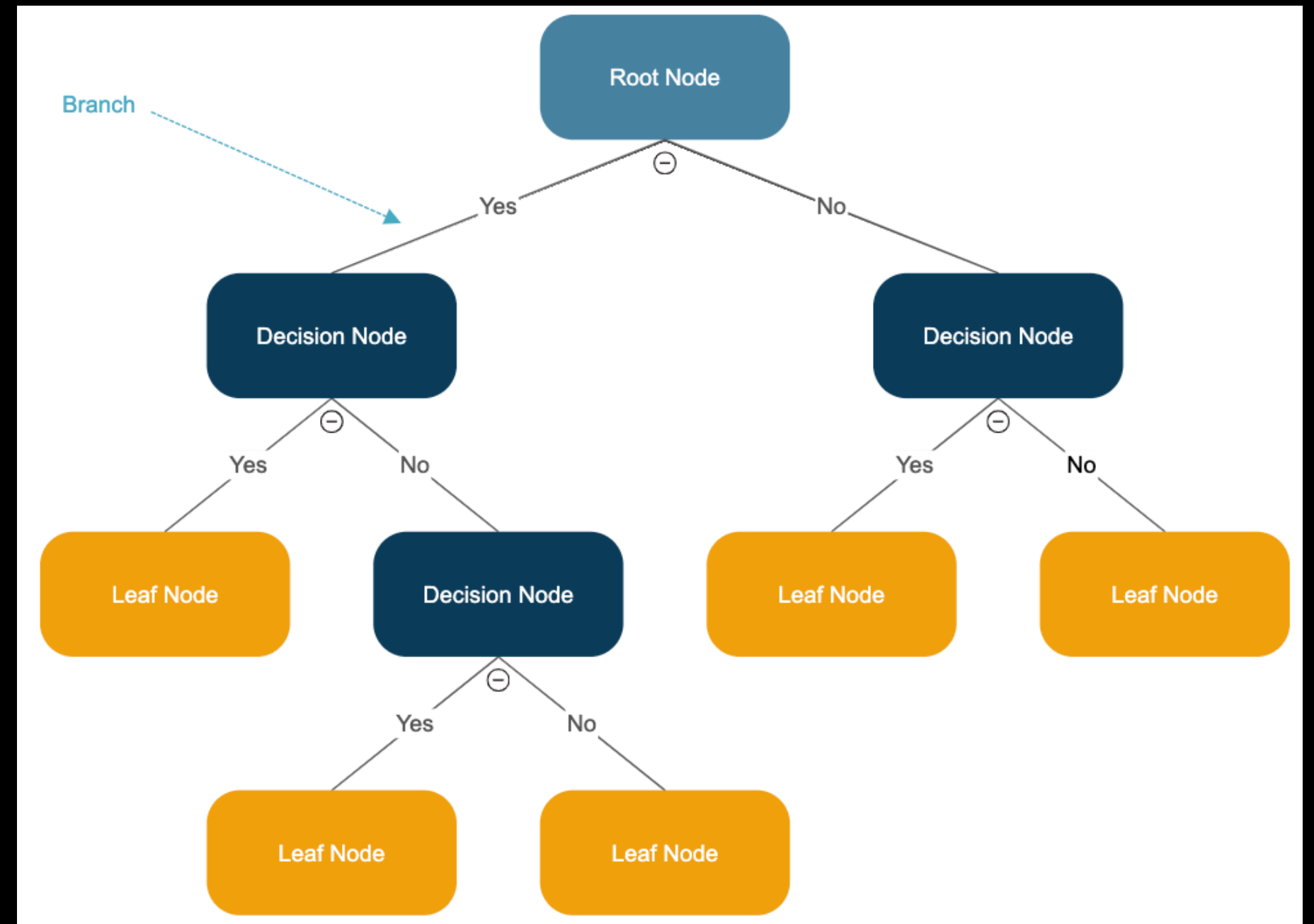
Objetivo

Nosso objetivo é treinar uma variável y , que representará as previsões do modelo, com o intuito de estimar de forma aproximada o preço real das residências. Para avaliar o desempenho do modelo, utilizaremos como métrica principal o MAE (Mean Absolute Error), que calcula os resíduos de cada previsão — diferenças absolutas entre os valores previstos e os valores reais — sem permitir que erros positivos e negativos se cancelem. A partir desses resíduos, determinamos a média, fornecendo uma medida clara do erro médio. Para alcançar essa meta, empregaremos dois algoritmos de aprendizado: Decision Tree e Random Forest. Após realizar os testes com ambos, analisaremos e compararemos seus desempenhos utilizando não apenas o MAE, mas também o MAPE (Mean Absolute Percentage Error). Esta métrica, expressa em porcentagem, oferece uma perspectiva intuitiva e facilmente compreensível, tornando os resultados mais acessíveis, inclusive para quem não possui familiaridade técnica com machine learning. Assim, poderemos comunicar as conclusões de maneira mais clara e impactante.



Decision Tree

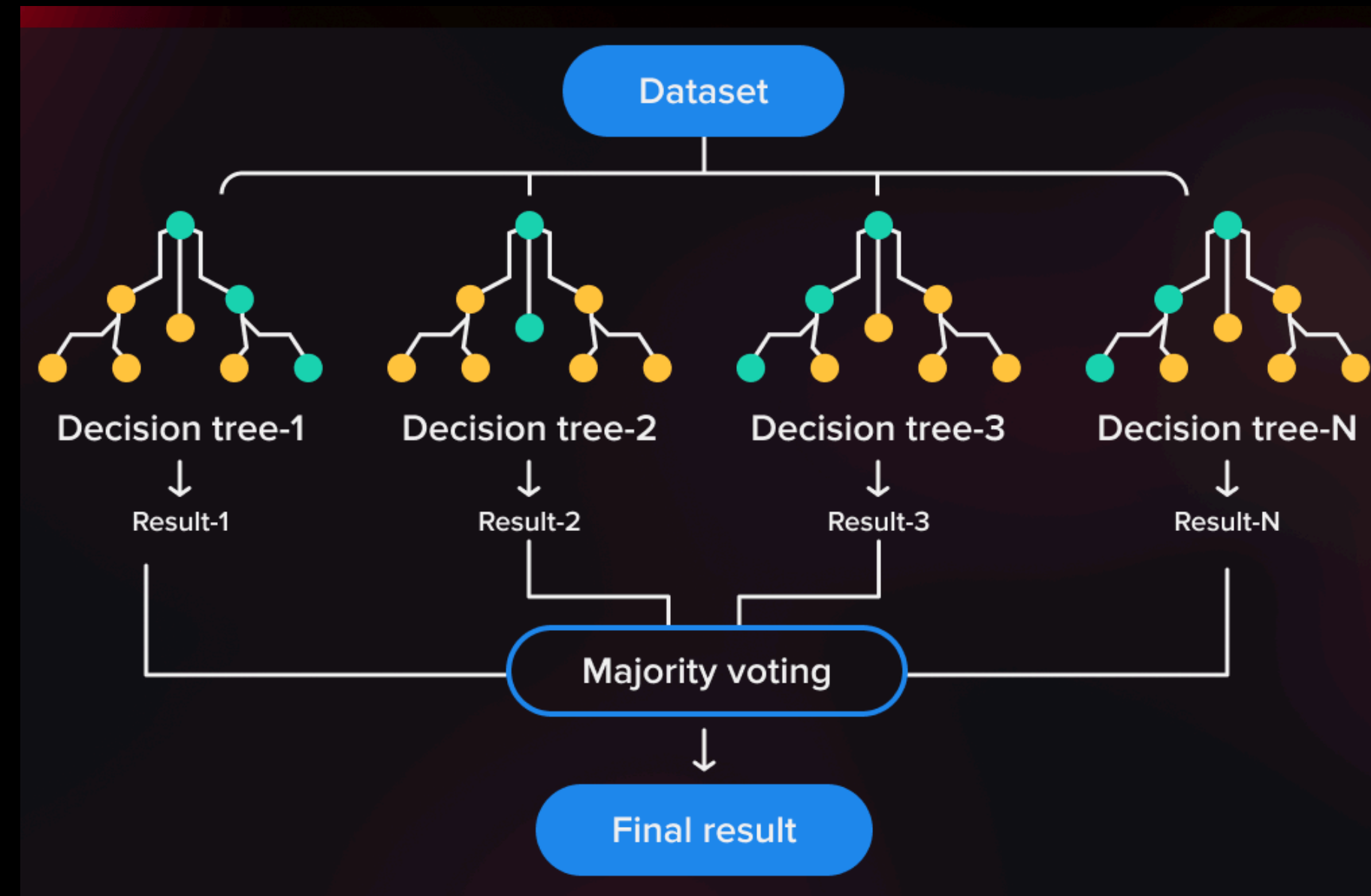
Uma Decision Tree, ou Árvore de Decisão, é um modelo de aprendizado de máquina amplamente utilizado tanto para problemas de classificação quanto de regressão. Ela organiza o processo de tomada de decisão em uma estrutura hierárquica, na qual as decisões são representadas por nós e ramos, formando uma estrutura semelhante a uma árvore. O modelo começa com um nó raiz, que representa o ponto inicial de análise, e, a partir dele, os dados são divididos de forma recursiva com base em condições aplicadas a suas variáveis. Cada divisão corresponde a uma decisão lógica, e o processo continua até que os dados sejam suficientemente particionados para realizar uma previsão.



Random Forest

O funcionamento do Random Forest envolve a criação de diversas árvores de decisão a partir de diferentes amostras do conjunto de dados, um processo conhecido como bootstrap. Em cada iteração, o modelo seleciona uma amostra aleatória com reposição dos dados originais, gerando conjuntos de dados ligeiramente diferentes para treinar cada árvore. Além disso, em cada nó de decisão, apenas um subconjunto aleatório de atributos é considerado para divisão, introduzindo mais aleatoriedade no processo e garantindo que as árvores sejam menos correlacionadas entre si.

A previsão final do Random Forest é obtida por meio de agregação. Para problemas de classificação, o modelo utiliza o voto majoritário das árvores; para regressão, calcula a média das previsões. Esse processo de combinação ajuda a suavizar os erros de árvores individuais, tornando o modelo mais robusto contra ruídos e variações nos dados.



Imports

Primeiramente, vamos importar algumas bibliotecas essenciais que serão a base para o funcionamento do nosso código, além de carregar os arquivos .csv que servirão como nosso dataset, trazendo os dados necessários para a análise e construção do modelo.

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

Importando o nosso DataFrame

```
[36] # Primeiro vamos contextualizar as informações do dataset
''' O conjunto de dados conhecido como California Housing reúne informações sobre os preços de imóveis por bairro no estado da Califórnia.
Entre os atributos disponíveis, estão a localização geográfica (longitude e latitude),
a média do número de quartos por residência, e o preço médio dos imóveis no bairro, que serve como rótulo'''

[38] # Usando housing_data como nome da variável pra ser menos genérico
#é mais significativo assim.

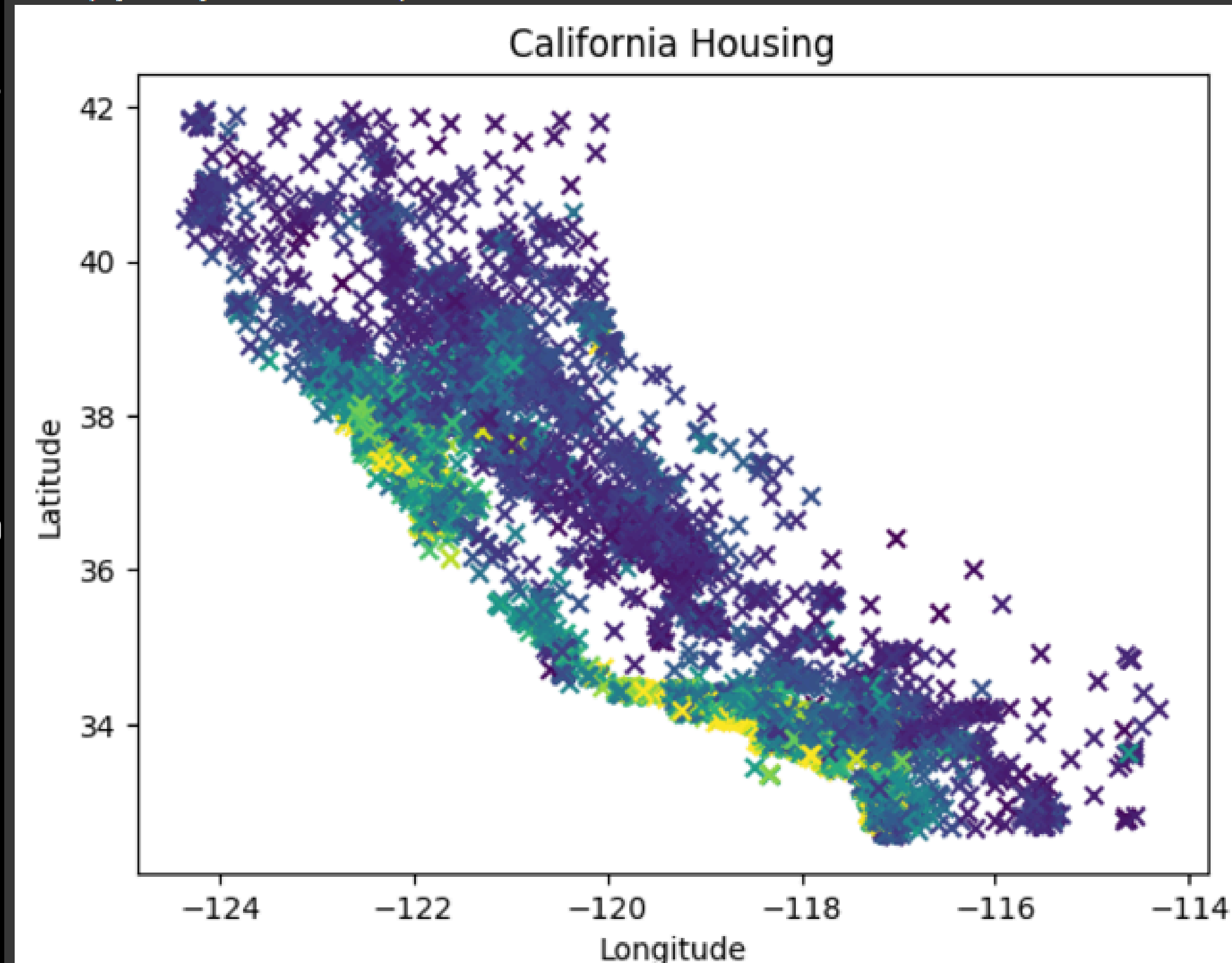
housing_data = pd.read_csv('/content/Housing.csv')
housing_data.describe()
```

Gráfico Geográfico


Neste gráfico, apresentamos a distribuição geográfica dos bairros representados no dataset. Observamos que, quanto mais próximos da região litorânea, mais elevados são os preços das residências. Isso evidencia que a localização geográfica desempenha um papel significativo na valorização dos imóveis, refletindo a influência do acesso à costa no mercado imobiliário.

```
# Por ser um estado litorâneo, geralmente, as residências mais próximas da praia são mais caras
# Como mostrado no gráfico abaixo, quanto mais próximo da praia, mais valiosa é a casa(verde e amarelo)
plt.scatter(dataset['longitude'], dataset['latitude'], marker='x', c=dataset['median_house_value'])
plt.title('California Housing')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
```

```
Text(0, 0.5, 'Latitude')
```



Treinando o Modelo

```
 #Criando uma função para achar o Mean Absolute Error

def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=0)
    model.fit(train_X, train_y)
    prediction_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, prediction_val)
    return mae
```

Função tem como objetivo calcular o Mean Absolute Error (MAE) para um modelo de árvore de decisão, considerando um número específico de nós-folha. Ela começa construindo um modelo de Decision Tree Regressor, ajustando-o aos dados de treinamento e utilizando o número máximo de nós-folha fornecido. Em seguida, o modelo treinado gera previsões a partir dos dados de validação. O MAE é então calculado, medindo a diferença média absoluta entre as previsões e os valores reais, fornecendo uma indicação precisa do desempenho do modelo. Ao final, a função retorna o MAE, permitindo avaliar a precisão do modelo para o número de nós-folha escolhido.

Treinando o modelo

Aqui é criado uma lista de parâmetros a fim de se obter uma comparação dos erros obtidos quando se muda as variáveis com as quais o modelo trabalha para fazer a predição.

```
PARAMETROS = {  
    ("yr_built", ): float("inf"),  
    ("bedrooms", "bathrooms"): float("inf"),  
    ("bedrooms", "bathrooms", "sqft_living"): float("inf"),  
    ("bedrooms", "bathrooms", "lat", "long"): float("inf"),  
    ('bedrooms', 'bathrooms', 'yr_built', 'grade', 'sqft_living', 'sqft_lot', 'sqft_above', 'floors', 'sqft_basement', 'sqft_living15', 'sqft_lot15'): float("inf"),  
    ('bedrooms', 'bathrooms', 'yr_renovated', 'grade', 'sqft_living', 'sqft_lot', 'sqft_above', 'floors', 'sqft_basement', 'sqft_living15', 'sqft_lot15'): float("inf"),  
}
```



EQUIPE 1

Treinando o modelo

O código realiza uma busca otimizada para determinar os melhores parâmetros de um modelo de árvore de decisão, com o objetivo de minimizar o erro absoluto médio (MAE). A cada iteração, ele trabalha com um conjunto específico de atributos, definidos previamente nos parâmetros, para ajustar os dados utilizados no treinamento e validação do modelo. Primeiro, a variável-alvo y é atribuída como o preço das casas, enquanto X contém apenas as características do conjunto atual de parâmetros. Em seguida, os dados são divididos em subconjuntos de treinamento e validação. Para cada conjunto de parâmetros, uma lista de possíveis valores para o número máximo de nós-folha é criada, abrangendo um intervalo especificado. O modelo então avalia cada valor candidato, calculando o erro para identificar qual configuração produz o menor MAE. Inicialmente, o erro é definido como infinito, e a cada iteração o menor erro encontrado substitui o anterior, registrando também o número de nós-folha que gerou o melhor desempenho. Ao final de cada conjunto de parâmetros, os resultados são exibidos, detalhando os atributos utilizados, o número ideal de nós-folha e o respectivo MAE, permitindo uma análise clara e comparativa do desempenho do modelo para diferentes combinações de variáveis.

```
for lista_de_parametros in PARAMETROS:

    # Criando o dado a ser previsto
    y = housing_data.price

    #Para cada iteração estamos mudando os parâmetros
    X = housing_data[[feature for feature in lista_de_parametros]]

    train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

    #Criamos uma lista com valores de max_leaf_nodes para serem comparados
    #Nosso objetivo é achar o valor que minimiza o erro
    lista_de_candidatos = list(range(10, 2_000, 5))

    #O erro inicialmente é um número muito grande
    erro = float("inf")

    for candidato in lista_de_candidatos:

        erro_do_candidato = get_mae(candidato, train_X, val_X, train_y, val_y)

        if erro_do_candidato < erro:
            erro = erro_do_candidato
            melhor_arvore = candidato

    print(lista_de_parametros, melhor_arvore, ":: MAE =", erro)
```

MAPE

Contextualizando: Vamos apresentar uma tabela comparando o valor real da casa, à esquerda, com a predição do modelo, à direita. Em seguida, calcular o MAPE.

```
print(np.vstack((y, predicted_home_prices)).T)
print(f'\nMAPE = {mean_absolute_percentage_error(y, predicted_home_prices)}')
```

```
[ [231300.      289499.23900879]
  [538000.      599728.09448819]
  [180000.      388470.88541667]
  ...
  [402101.      332630.73300971]
  [400000.      435715.54595444]
  [325000.      332630.73300971]]
```

```
MAPE = 0.2440503929703485
```


Teste 1: Decision Tree 340 Leafs

```
#Criando os dados de features que gerou o segundo menor erro

features = ['bedrooms', 'bathrooms', 'yr_built', 'grade', 'sqft_living', 'sqft_lot', 'sqft_above', 'floors', 'sqft_basement', 'sqft_living15', 'sqft_lot15']

X = housing_data[features]

#Aplicando o modelo na totalidade dos dados

modelo_final_dt = DecisionTreeRegressor(max_leaf_nodes=340)
modelo_final_dt.fit(X, y)

predicted_home_prices = modelo_final_dt.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

→ 108680.4581266811

Vamos apresentar uma tabela comparando o valor real da casa, à esquerda, com a predição do modelo, à direita.

```
[51] print(np.vstack((y, predicted_home_prices)).T)
      print(f'\nMAPE = {mean_absolute_percentage_error(y, predicted_home_prices)}')
```

→

[231300.	289499.23900879]
[538000.	599728.09448819]
[180000.	388470.88541667]
...	
[402101.	332630.73300971]
[400000.	435715.54595444]
[325000.	332630.73300971]

MAPE = 0.2440503929703485

Teste 2: Decision Tree 950 Leafs

Ao utilizar o modelo de Decision Tree, observa-se que tanto o MAE quanto o MAPE apresentam um desempenho mais eficiente à medida que o número de nós-folha (`max_leaf_nodes`) é incrementado, indicando que uma maior complexidade da árvore contribui para previsões mais precisas e uma melhor capacidade de capturar os padrões subjacentes dos dados.



```
#Criando os dados de features que gerou o menor erro
```

```
features = ["bedrooms", "bathrooms", "lat", "long"]
```

```
#Aplicando o modelo na totalidade dos dados
```

```
modelo_final_dt = DecisionTreeRegressor(max_leaf_nodes=950)
```

```
modelo_final_dt.fit(X, y)
```

```
predicted_home_prices = modelo_final_dt.predict(X)
```

```
mean_absolute_error(y, predicted_home_prices)
```



```
88909.65128082618
```

```
[53]
```

```
print(np.vstack((y, predicted_home_prices)).T)
```

```
print(f'\nMAPE = {mean_absolute_percentage_error(y, predicted_home_prices)}
```



```
[[231300.      274786.07451565]
 [538000.      685208.49056604]
 [180000.      312674.44444444]
 ...
 [402101.      320054.28834356]
 [400000.      455526.63905325]
 [325000.      320054.28834356]]
```

```
MAPE = 0.20928534617339614
```

Teste 3: Random Forest

Ao aplicar o algoritmo Random Forest, percebemos que seu desempenho supera significativamente o do Decision Tree, demonstrando maior eficiência e precisão nas previsões. Assim, para otimizar o modelo e alcançar resultados superiores, o uso do RandomForestRegressor se mostra a escolha mais apropriada. Contudo, ainda é possível refinar o algoritmo, ajustando seus hiperparâmetros para torná-lo ainda mais eficaz, como será demonstrado no teste 4 a seguir, onde buscamos maximizar sua precisão e desempenho.

```
#Criando os dados de features que gerou o segundo menor erro

features = ['bedrooms', 'bathrooms', 'yr_built', 'grade', 'sqft_living', 'sqft_lot', 'sqft_above', 'floors', 'sqft_basement', 'sqft_living15', 'sqft_lot15']

X = housing_data[features]

#Aplicando o modelo na totalidade dos dados

modelo_final_rf = RandomForestRegressor()
modelo_final_rf.fit(X, y)

predicted_home_prices = modelo_final_rf.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

⇒ 43922.31560691794

```
[29] print(np.vstack((y, predicted_home_prices)).T)
print(f'\nMAPE = {mean_absolute_percentage_error(y, predicted_home_prices)}')
```

⇒

[231300.	266804.5]
[538000.	610095.47]
[180000.	255045.8]
...	
[402101.	380603.72]
[400000.	397805.]
[325000.	345829.6]]

MAPE = 0.08892360114552567

Teste 4: Random Forest Aplicação Final

Nesta última aplicação, exploramos ao máximo o potencial do modelo ao utilizar todos os parâmetros disponíveis no dataset. Ao incluir a totalidade das variáveis, permitimos que o algoritmo capturasse as interações mais complexas e os padrões mais sutis nos dados, o que resultou em um desempenho significativamente superior. Essa abordagem levou à obtenção do menor MAE e do menor MAPE dentre todos os testes realizados, demonstrando a importância de trabalhar com uma ampla gama de atributos relevantes para melhorar a precisão das previsões. Este resultado reforça que, quando adequadamente configurado, o modelo é capaz de atingir altos níveis de eficiência e confiabilidade, proporcionando previsões mais próximas da realidade e métricas de erro consideravelmente reduzidas.

```
features = ['bedrooms', 'bathrooms', 'yr_built', 'grade', 'sqft_living', 'sqft_lot', 'sqft_above', 'floors', 'sqft_basement', 'sqft_living15', 'sqft_lot15', "waterfront", "view", "condition", "yr_renovated"]

X = housing_data[features]

#Aplicando o modelo na totalidade dos dados

modelo_final_rf = RandomForestRegressor()
modelo_final_rf.fit(X, y)

predicted_home_prices = modelo_final_rf.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

⇒ 25786.35114078057

```
[33] print(np.vstack((y, predicted_home_prices)).T)
      print(f'\nMAPE = {mean_absolute_percentage_error(y, predicted_home_prices)}')
```

⇒

[231300.	238375.58]
[538000.	500927.57666667]	
[180000.	217819.45]
...		
[402101.	374230.95]
[400000.	403768.13]
[325000.	336741.59]]

Teste dos Algoritmos Utilizados

TESTE 1

Decision Treee com
(max_leaf_nodes=340)
MAE: 108680.4581266811
MAPE = 0.2440503929703485

TESTE 2

Decision Tree com
(max_leaf_nodes=950).
MAE: 88909.65128082618
MAPE = 0.20928534617339614

TESTE 3

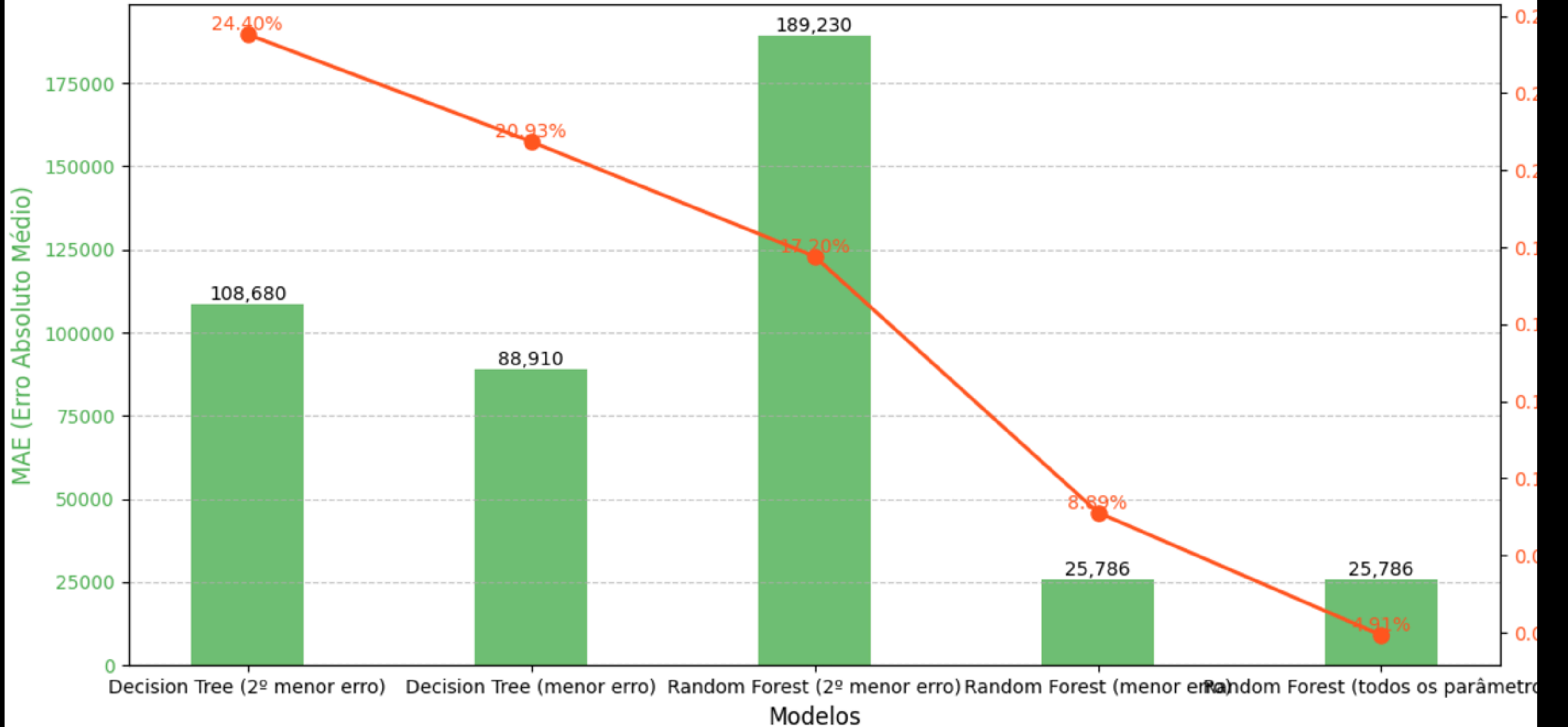
Random Forest com
todos os parâmetros
MAE: 43922.31560691794
0.08892360114552567

TESTE 4

Random Forest com
todos os parâmetros
MAE: 25786.35114078057
MAPE = 0.04909743252819053



Impacto do Ajuste de Parâmetros no Desempenho dos Modelos



Algoritmo com Bayesian Regression

Autor: Andrey Shtrauss

$$y \sim N(\beta^T X, \sigma^2 I)$$

A regressão linear univariada Bayesiana é uma abordagem de Regressão Linear onde a análise estatística é realizada dentro do contexto de inferência Bayesiana. A Regressão Bayesiana ajuda a fazer previsões que não só levam em conta os dados, mas também as incertezas e as crenças iniciais. É como fazer previsões com um pouco mais de cautela e informação, o que pode ser especialmente útil em situações complexas ou com muitos fatores desconhecidos. Se o dataset California Housing tiver um número moderado de amostras e atributos, a Bayesian Regression pode ser mais eficiente e produzir previsões mais confiáveis. O Random Forest, por outro lado, tende a se sair melhor em datasets maiores ou com relações complexas e altamente não lineares entre as variáveis. Elas também permitem não apenas prever o preço esperado, mas também avaliar a probabilidade de preços fora de um intervalo esperado, algo que a gente somente usando Random Forest não poderemos conseguir. Por isso, o algoritmo do Andrey é melhor que o nosso.

```
modelEval(trdata_upd,model_id='rf')
```

```
Scores: [48078.93558134 48376.18009495 48989.2628633 47516.84556066  
48382.16223432]  
Mean: 48268.6772669145  
std: 478.5252593847063
```

- We can see that even a very powerful like RandomForest () can't really reduce the RMSE with the current set of feature instance combination very much further (despite being a model that very easily can overpredict)
- An error of 48,000 is quite a large in the context of the target values & it's probably a reasonable idea to investigate validity of the current dataset & review it
- The dataset is quite old & doesn't contain the best features compared to more recent datasets, like the introductory [Price Problem](#) (which was quite fun!), but hopefully the basic approach one can take was clear.

<https://www.kaggle.com/code/shtrausslearning/bayesian-regression-house-price-prediction>

Vitor

Conclusão

Concluindo, para garantir que nosso código atinja o máximo de eficiência e precisão, a escolha ideal recai sobre o algoritmo Random Forest. Essa abordagem nos permite alcançar os menores valores de MAE e MAPE, especialmente ao utilizar todos os parâmetros disponíveis no dataset. Essa escolha não apenas aumenta a qualidade das previsões, aproximando-as do valor real dos preços das residências, mas também evita problemas comuns, como o overfitting, graças à robustez intrínseca do modelo. Dessa forma, conseguimos equilibrar precisão e generalização, garantindo um desempenho confiável mesmo em cenários variados.

Além disso, essa etapa do processo seletivo foi mais do que uma aplicação técnica; ela proporcionou uma introdução prática e significativa ao universo do machine learning, ampliando nossa compreensão sobre os fundamentos. A plataforma Kaggle desempenhou um papel essencial nesse aprendizado, servindo como uma ferramenta indispensável durante todo o trabalho. Ela nos permitiu explorar dados reais, experimentar diferentes abordagens e entender melhor as nuances que envolvem a criação de algoritmos eficientes. Assim, não apenas aprimoramos nosso modelo, mas também consolidamos uma base sólida para futuras aplicações e projetos na área de ciência de dados.