RELATÓRIO – MINERAÇÃO DE PREÇOS DE CASAS

Código:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

dados = pd.read_csv("/content/PRECOS_CASAS.csv")
```

Explicação:

. Importação da biblioteca pandas:

Esta linha importa a biblioteca pandas e a atribui um apelido `pd`. Esse apelido permite que você use `pd` em vez do nome completo da biblioteca `pandas` em todo o seu código. Pandas é uma biblioteca Python poderosa para análise e manipulação de dados.

2. Leitura de dados do arquivo CSV:

Esta linha lê dados de um arquivo CSV chamado "PRECOS_CASAS.csv" localizado no diretório "/content" e os atribui a uma variável chamada `dados`. Assume-se que o arquivo CSV seja separado por vírgulas (o delimitador padrão para pandas.read_csv).

Em resumo, este trecho de código importa a biblioteca pandas como `pd` e lê um arquivo CSV chamado "PRECOS_CASAS.csv" em um objeto DataFrame chamado `dados`.

código:

```
dados.isnull().sum()
dados.dropna(inplace=True)
```

Explicação:

1. Identificação de valores ausentes:

- **A linha `dados.isnull().sum()` identifica e conta a quantidade de valores ausentes (NaN) em cada coluna do DataFrame `dados`.**

2. Remoção de linhas com valores ausentes:

- **A linha `dados.dropna(inplace=True)` remove as linhas do DataFrame `dados` que contêm pelo menos um valor ausente em qualquer coluna. As modificações são feitas no próprio DataFrame `dados`.**

Código:

```
scaler = StandardScaler()

dados["Area"] = scaler.fit_transform(dados["Area"].values.reshape(-1, 1))
dados["Quartos"] = scaler.fit_transform(dados["Quartos"].values.reshape(-1, 1))
dados["Banheiros"] = scaler.fit_transform(dados["Banheiros"].values.reshape(-1, 1))
```

Explicação:

1. **Criar um objeto `StandardScaler` para padronização:**
    - `scaler = StandardScaler()`
2. **Padronizar cada coluna numérica:**
    - Para cada coluna (por exemplo, "Area"):
        - Selecionar a coluna: `dados["Area"]`
        - Converter para array NumPy: `.values`
        - Reformatar o array: `.reshape(-1, 1)`
        - Padronizar e armazenar: `dados["Area"] = scaler.fit_transform(...)`

Explicação das etapas:

1. Criando o `StandardScaler`:

- O objeto `StandardScaler` é criado para realizar a padronização dos dados.
- Ele ajusta os valores para uma média de 0 e um desvio padrão de 1, garantindo que todas as features tenham a mesma escala.

2. Padronizando as colunas:

- O loop percorre cada coluna numérica ("Area", "Quartos", "Banheiros").
- Para cada coluna:

- A coluna é selecionada do DataFrame.
- Os valores da coluna são convertidos em um array NumPy para facilitar a manipulação.
- O array é redimensionado para o formato compatível com o `fit_transform`.
- A padronização é aplicada usando o `fit_transform` do `StandardScaler`.
- O resultado da padronização (novo array) é armazenado de volta na coluna original do DataFrame, sobrescrevendo os valores originais com os padronizados.

Código:

```
X = dados[["Area", "Quartos", "Banheiros"]]

y = dados["Preco"]



X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
random_state=42)
```

Explicação:

Padronização das colunas numéricas:

- As colunas "Area", "Quartos" e "Banheiros" são padronizadas para uma escala comum com média 0 e desvio padrão 1 (linhas 1-4).

Criação dos conjuntos de dados para machine learning:

- O DataFrame `dados` é dividido em dois conjuntos de dados:
  - `X`: Contém as colunas de features ("Area", "Quartos", "Banheiros") após a padronização (linha 5).
  - `y`: Contém a coluna de target ("Preco") (linha 6).

Divisão do conjunto de dados em treinamento e teste:

- O conjunto de dados `X` e `y` é dividido em conjuntos de treinamento (`X_train`, `y_train`) e teste (`X_test`, `y_test`) (linha 7).
  - 80% dos dados são utilizados para treinamento.
  - 20% dos dados são utilizados para teste.
  - Um estado inicial aleatório (`random_state=42`) é definido para garantir reprodutibilidade.

Código:

```
modelo = LinearRegression()
modelo.fit(X_train, y_train)
```

Explicação:

Importação da biblioteca scikit-learn:

- A biblioteca `sklearn` é utilizada para machine learning, incluindo a classe `LinearRegression` para criar modelos de regressão linear.

Criação do modelo de regressão linear:

- Um objeto `modelo` da classe `LinearRegression` é criado.

Treinamento do modelo:

- O modelo é treinado utilizando o conjunto de treinamento `X_train` (features) e `y_train` (target).
- Durante o treinamento, o modelo aprende os parâmetros da reta de regressão que melhor se ajustam aos dados.

Código:

```
y_pred = modelo.predict(X_test)



rmse = mean_squared_error(y_test, y_pred)

print(f"RMSE: {rmse:.2f}")



r2 = r2_score(y_test, y_pred)

print(f"R²: {r2:.2f}")
```

Explicação:

1. **Prever preços em dados de teste:**
    - O modelo treinado é utilizado para fazer previsões de preços em novos dados de teste ("X_test").
2. **Calcular o RMSE:**
    - A diferença entre os valores reais e previstos é calculada e utilizada para calcular o RMSE.
3. **Calcular o R²:**
    - O R² é calculado com base na relação entre os valores reais e previstos.
4. **Imprimir os resultados:**

- Os valores do RMSE e R^2 são impressos para avaliar o desempenho do modelo.

Interpretação:

- Um RMSE baixo e um R^2 alto indicam que o modelo está performando bem e generalizando bem para novos dados.
- Outros fatores podem ser considerados para avaliar o modelo, como a distribuição dos erros e a visualização dos resultados.

Código:

```
print(f"Coeficientes: {modelo.coef_}")
```

Explicação:

Exibir os coeficientes do modelo de regressão linear treinado para prever o preço de casas ("Preco") com base em suas características ("Area", "Quartos", "Banheiros").

Interpretação dos Coeficientes:

- Cada coeficiente representa a mudança no target ("Preco") esperada para uma mudança unitária na feature correspondente, mantendo as outras features constantes.
- Um coeficiente positivo indica uma relação direta entre a feature e o target, enquanto um coeficiente negativo indica uma relação inversa.
- A magnitude do coeficiente indica a força da relação entre a feature e o target.

Código:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Explicação:

matplotlib.pyplot (plt):

- Utilizada para criar gráficos e visualizações de dados.
- Oferece funções para criar figuras, eixos, gráficos e outros elementos de visualização.

seaborn (sns):

- Baseada no matplotlib.
- Fornece estilos predefinidos e funções de alto nível para criar visualizações estatísticas.
- Simplifica a criação de gráficos com visualizações complexas.

Código:

```
plt.figure(figsize=(8, 6))

sns.distplot(dados["Preco"])

plt.xlabel("Preco")

plt.ylabel("Densidade")

plt.title("Distribuição de Preços das Casas")

plt.show()
```

Explicação:

Objetivo:

Criar um gráfico de distribuição para visualizar a frequência dos diferentes valores de preço das casas em um conjunto de dados.

Etapas:

1. Configurar o tamanho da figura (8 polegadas de largura, 6 polegadas de altura).
2. Criar o gráfico de distribuição usando a função `sns.distplot` do `seaborn` para os preços das casas.
3. Definir rótulos dos eixos: "Preco" para o eixo X e "Densidade" para o eixo Y.
4. Definir título do gráfico: "Distribuição de Preços das Casas".
5. Exibir o gráfico na tela.

Código:

```
plt.figure(figsize=(10, 6))

sns.heatmap(dados[["Area", "Quartos", "Banheiros", "Preco"]].corr(), annot=True)

plt.title("Mapa de Calor: Correlação entre Variáveis")

plt.show()
```

Explicação:

Objetivo:

Criar um heatmap para visualizar as correlações entre as variáveis em um conjunto de dados.

Etapas:

1. Configurar o tamanho da figura (10 polegadas de largura, 6 polegadas de altura).
2. Criar o heatmap de correlação:
    - Calcular a matriz de correlação (correlação de Pearson) entre as variáveis.
    - Usar o `seaborn` para criar o heatmap a partir da matriz de correlação.
    - Anotar os valores de correlação no heatmap.
3. Definir título do gráfico: "Mapa de Calor: Correlação entre Variáveis".
4. Exibir o heatmap na tela.