
MVP: Análise Exploratória e Pré-Processamento de Dados

Dados do Aluno

- **Nome:** Denilson Santos
 - **Disciplina:** Análise Exploratória e Pré-Processamento de Dados
 - **Pós-Graduação:** Ciência de dados e Analytics
-

✓ 1. Origem dos Dados

O Dataset utilizado é referente aos Aluguéis de imóveis no Brasil. O dataset utilizado foi encontrado no site do kaggle, no seguinte endereço:

<https://www.kaggle.com/datasets/rubenssjr/brasilian-houses-to-rent?resource=download>

Dados Utilizados

- **city:** Cidade onde o imóvel está localizado
- **area:** Área total do imóvel
- **rooms:** Quantidade de Quartos
- **bathroom:** Quantidade de Banheiros
- **parking_spaces:** Quantidade de Vagas de Estacionamento
- **floor:** Quantidade de pisos (andar) do imóvel
- **animal:** Se aceita pets ou não
- **furniture:** Se o imóvel está com móveis ou não
- **hoa_brl:** é uma taxa que proprietários pagam para custear a manutenção de áreas comuns dos condomínios.
- **rent_amount:** Valor do Aluguel
- **property_tax:** Contribuição predial
- **fire_insurance:** Taxa de Incêndio
- **total:** Valor total do Aluguel

2. Definição do Problema

A diretoria da Imobiliária Luxor solicitou que a equipe de dados faça uma análise dos Aluguéis no Brasil, com foco na cidade do Rio de Janeiro para futura expansão da empresa.

A equipe decidiu fazer uma análise exploratória dos dados e depois utilizar alguns modelos de Machine Learning para avaliar qual seria a melhor solução. Serão utilizados os seguintes modelos:

- Regressão Linear
- Regressão Logística
- Árvore de Decisão
- Redes Neurais

✓ 3. Preparação dos Dados

Nesta etapa, iremos importar as bibliotecas necessárias e importar o Dataset que será utilizado.

```
1 # Importando as bibliotecas necessárias
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import seaborn as sns
6 import io
7 import statsmodels.api as sm
8
9 from scipy import stats
10 from sklearn.preprocessing import StandardScaler, OneHotEncoder
11 from sklearn.compose import ColumnTransformer
12 from sklearn.pipeline import Pipeline
13 from sklearn.ensemble import RandomForestClassifier
14 from sklearn.metrics import classification_report
15 from sklearn.model_selection import GridSearchCV
16 from sklearn.model_selection import train_test_split
17 from sklearn.model_selection import cross_val_score
18 from sklearn.model_selection import KFold
19 from sklearn.metrics import mean_squared_error, r2_score
20
21 from tabulate import tabulate
22 from datetime import datetime
23 from csv import DictReader
24
25 from google.colab import data_table
26 data_table.enable_dataframe_formatter()
```

Importação dos Dados

```
1 # Dataset:
2 url = "https://raw.githubusercontent.com/CITMAX/data/main/houses_to_rent_complete.csv"
3 df = pd.read_csv(url, sep=',', on_bad_lines='warn')
```

Realizando a leitura e teste dos dados

1 df

1 to 25 of 10692 entries

Filter

index	city	area	rooms	bathroom	parking spaces	floor	animal	furniture	hoa (R\$)	ren
0	São Paulo	70	2	1	1	7	accept	furnished	2065	
1	São Paulo	320	4	4	0	20	accept	not furnished	1200	
2	Porto Alegre	80	1	1	1	6	accept	not furnished	1000	
3	Porto Alegre	51	2	1	0	2	accept	not furnished	270	
4	São Paulo	25	1	1	0	1	not accept	not furnished	0	
5	São Paulo	376	3	3	7	-	accept	not furnished	0	
6	Rio de Janeiro	72	2	1	0	7	accept	not furnished	740	
7	São Paulo	213	4	4	4	4	accept	not furnished	2254	
8	São Paulo	152	2	2	1	3	accept	furnished	1000	
9	Rio de Janeiro	35	1	1	0	2	accept	furnished	590	
10	São Paulo	26	1	1	0	2	accept	furnished	470	
11	Campinas	46	1	1	1	10	accept	not furnished	550	
12	São Paulo	36	1	1	0	11	accept	not furnished	359	
13	São Paulo	55	1	1	1	2	accept	furnished	790	
14	São Paulo	100	2	2	2	24	accept	furnished	900	
15	Campinas	330	4	6	6	-	accept	furnished	680	
16	São Paulo	110	2	2	1	1	accept	not furnished	700	

Next steps:

Generate code with df



View recommended plots

New interactive sheet

✓ 4 - Análise Exploratória

Neste tópico, iremos explorar os dados, utilizando algumas análises.

```
1 # Filtrar apenas pela cidade do Rio de Janeiro
2 df = df[df['city'] == 'Rio de Janeiro']
```

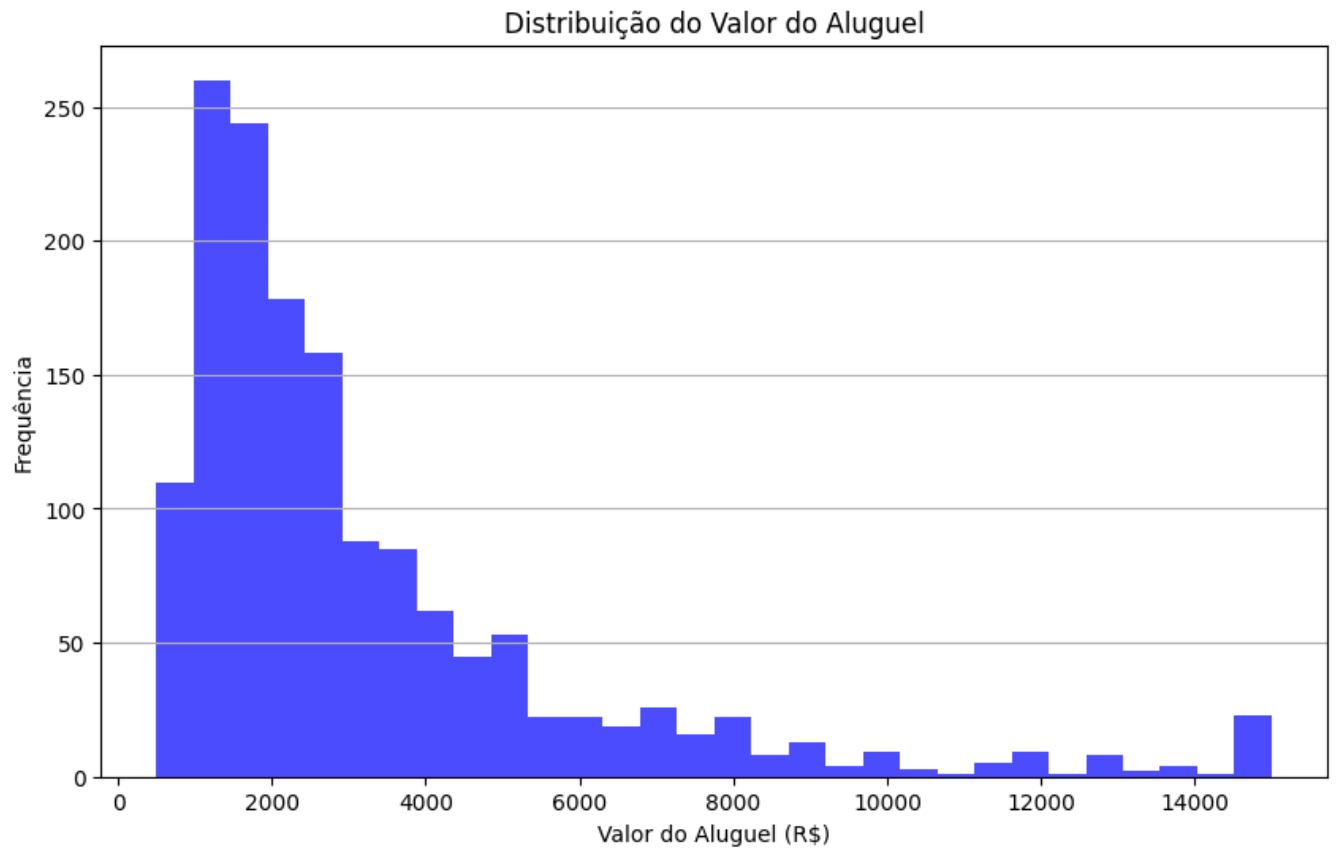
4.1. Visualização da Distribuição dos Valores de Aluguel

Um histograma pode ajudar a visualizar a distribuição do valor do aluguel.

```

1 import matplotlib.pyplot as plt
2
3 # Histograma do valor do aluguel
4 plt.figure(figsize=(10, 6))
5 plt.hist(df['rent amount (R$)'], bins=30, color='blue', alpha=0.7)
6 plt.title('Distribuição do Valor do Aluguel')
7 plt.xlabel('Valor do Aluguel (R$)')
8 plt.ylabel('Frequência')
9 plt.grid(axis='y')
10 plt.show()

```



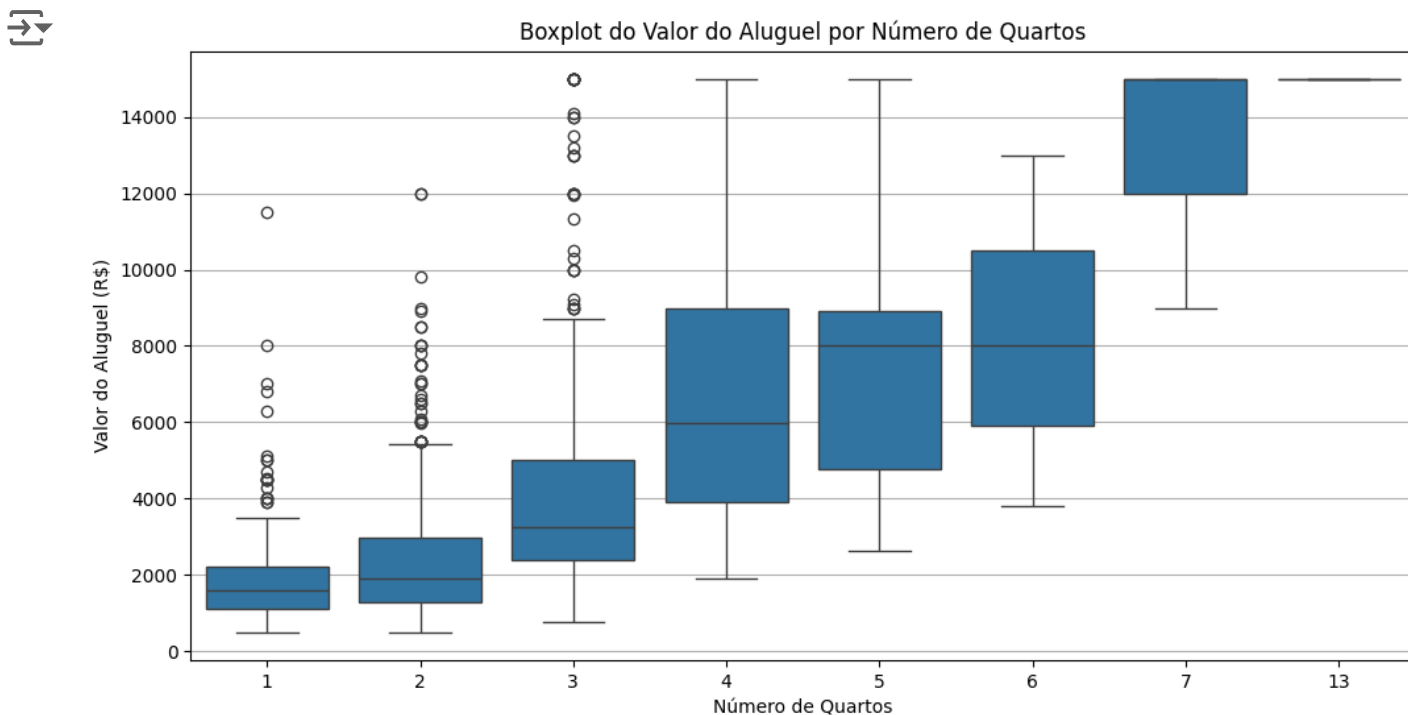
4.2. Gráfico de Boxplot para Analisar Outliers

Um boxplot pode ser utilizado para identificar outliers nos valores de aluguel.

```

1 import seaborn as sns
2
3 # Boxplot do valor do aluguel por número de quartos
4 plt.figure(figsize=(12, 6))
5 sns.boxplot(x='rooms', y='rent amount (R$)', data=df)
6 plt.title('Boxplot do Valor do Aluguel por Número de Quartos')
7 plt.xlabel('Número de Quartos')
8 plt.ylabel('Valor do Aluguel (R$)')
9 plt.grid(axis='y')
10 plt.show()

```



4.3. Matriz de Correlação

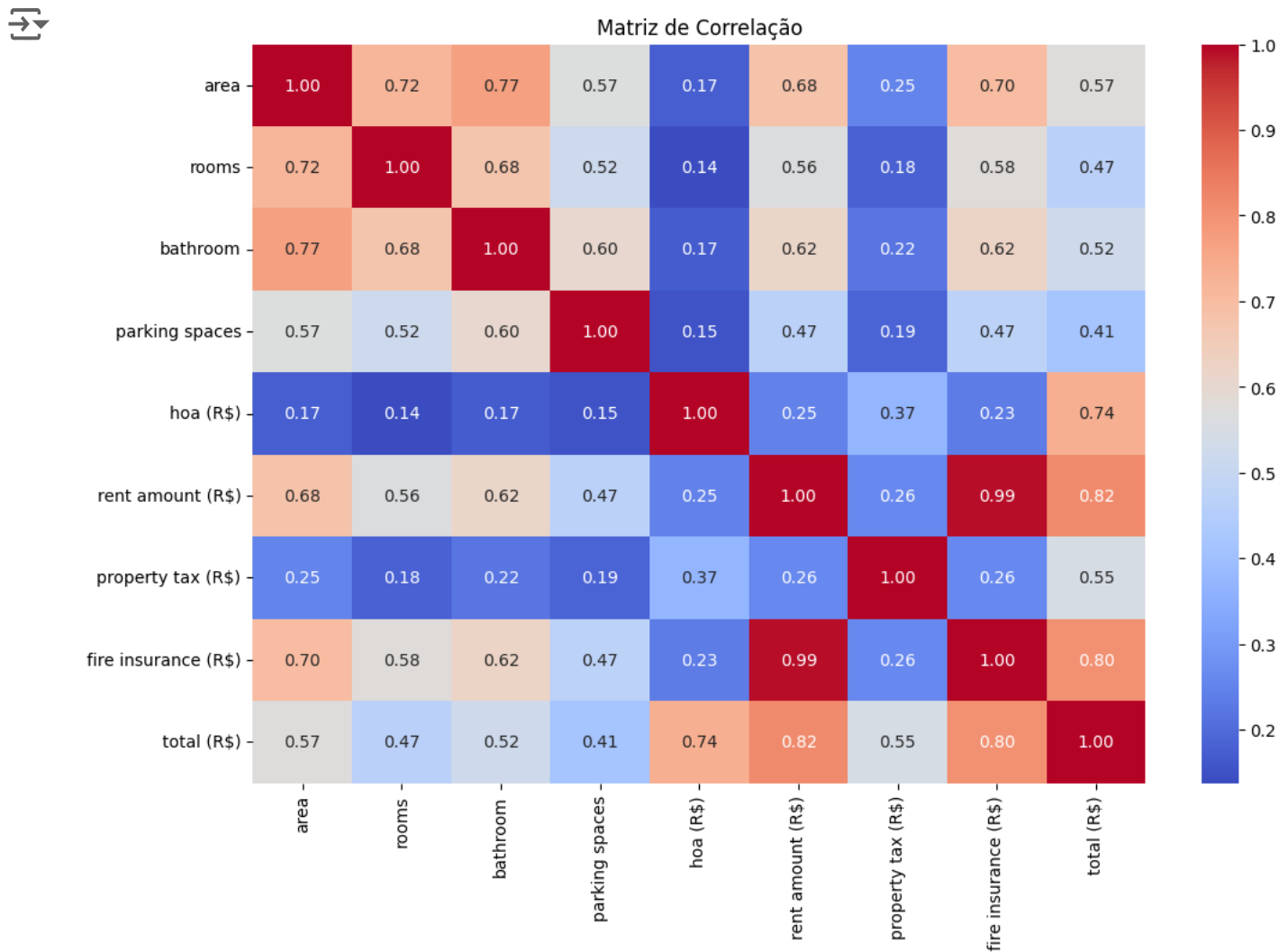
A matriz de correlação ajuda a entender como as variáveis estão relacionadas entre si.

```

1 # Matriz de correlação apenas com colunas numéricas
2 correlation_matrix = df.corr(numeric_only=True)
3
4 # Gráfico de calor da matriz de correlação
5 plt.figure(figsize=(12, 8))
6 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
7 plt.title('Matriz de Correlação')

```

```
8 plt.show()
```



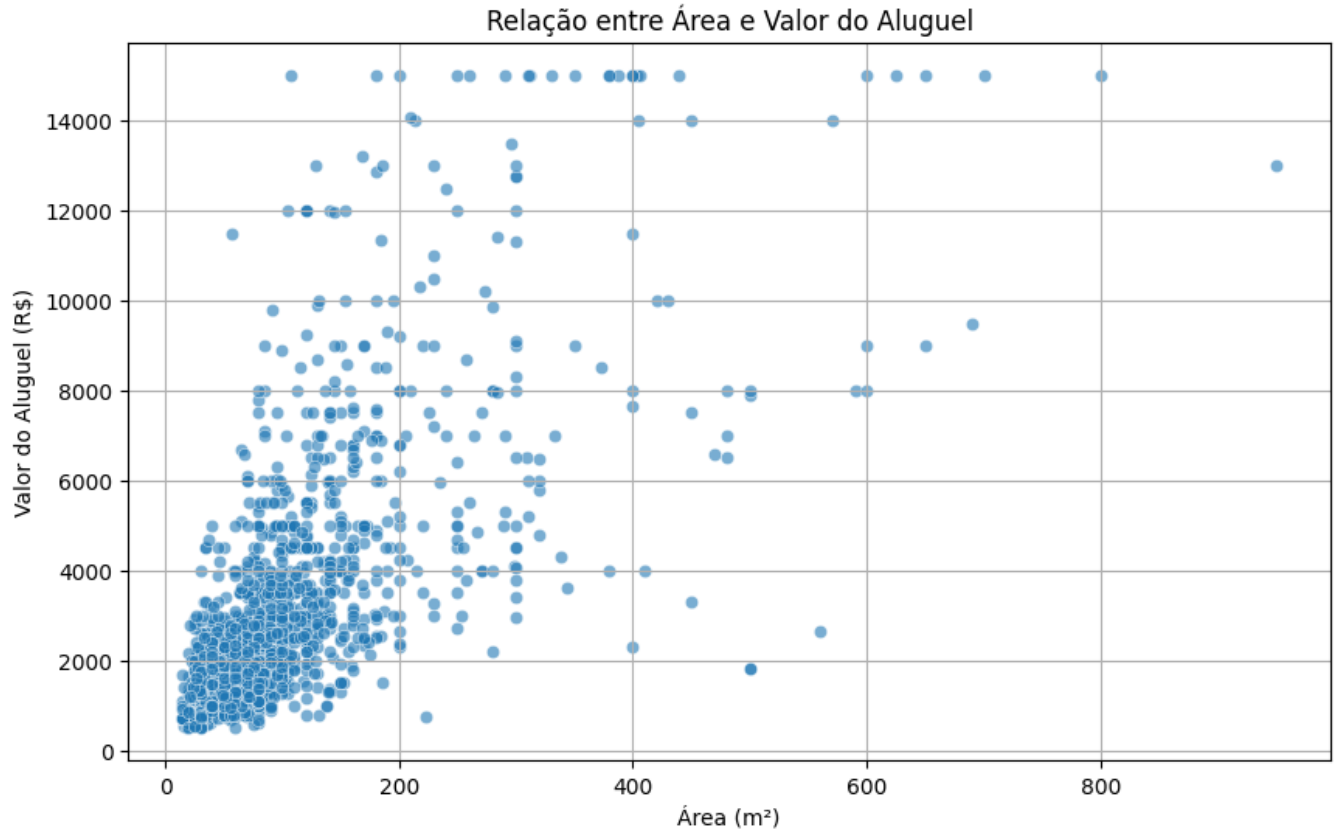
4.4. Análise da Relação entre Área e Valor do Aluguel

Um gráfico de dispersão pode ser útil para visualizar a relação entre a área e o valor do aluguel.

```

1 # Gráfico de dispersão entre área e valor do aluguel
2 plt.figure(figsize=(10, 6))
3 sns.scatterplot(x='area', y='rent amount (R$)', data=df, alpha=0.6)
4 plt.title('Relação entre Área e Valor do Aluguel')
5 plt.xlabel('Área (m²)')
6 plt.ylabel('Valor do Aluguel (R$)')
7 plt.grid()
8 plt.show()

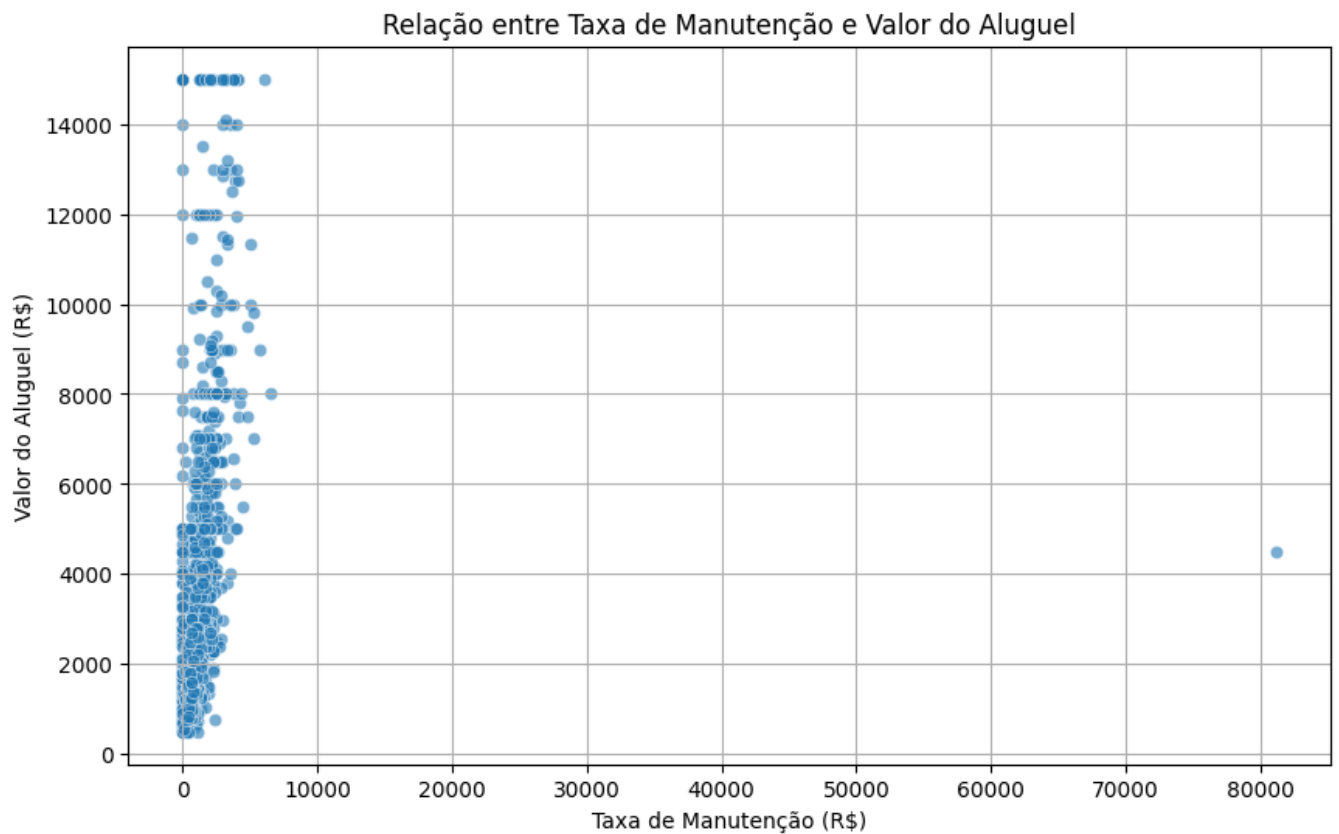
```



4.5. Análise da Taxa de Manutenção em Relação ao Valor do Aluguel

Verificar como a taxa de manutenção (hoa_brl) se relaciona com o valor do aluguel.


```
1 # Gráfico de dispersão entre hoa_brl e valor do aluguel
2 plt.figure(figsize=(10, 6))
3 sns.scatterplot(x='hoa (R$)', y='rent amount (R$)', data=df, alpha=0.6)
4 plt.title('Relação entre Taxa de Manutenção e Valor do Aluguel')
5 plt.xlabel('Taxa de Manutenção (R$)')
6 plt.ylabel('Valor do Aluguel (R$)')
7 plt.grid()
8 plt.show()
```



✓ 5. Modelos Supervisionados

5.1 Tratamento dos Dados

Nesta etapa, realizar o tratamento dos dados

```
1 # Filtrar apenas pela cidade do Rio de Janeiro
2 df = df[df['city'] == 'Rio de Janeiro']
```

```
1 # Verificar a distribuição de valores nulos
2 print(f"Valores Nulos: {df.isnull().sum()}")
```

```
⇒ Valores Nulos: city          0
   area                      0
   rooms                     0
   bathroom                  0
   parking spaces            0
   floor                     0
   animal                    0
   furniture                  0
   hoa (R$)                   0
   rent amount (R$)          0
   property tax (R$)         0
   fire insurance (R$)       0
   total (R$)                 0
   dtype: int64
```

```
1 # Verificar a presença de valores duplicados
2 print(f"Valores Duplicados: {df.duplicated().sum()}")
```

```
⇒ Valores Duplicados: 70
```

```
1 # Remover duplicatas com base em colunas específicas
2 df = df.drop_duplicates(subset=['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'rent amount (R$)'])
3
4 print(f"Valores Duplicados Após Remoção: {df.duplicated().sum()}")
5
6 print(f"Número de linhas antes da filtragem: {len(df)}")
7 df = df[df['hoa (R$)'] <= 1.5 * df['rent amount (R$)']]
8 print(f"Número de linhas após a filtragem: {len(df)}")
```

```
⇒ Valores Duplicados Após Remoção: 0
   Número de linhas antes da filtragem: 1424
   Número de linhas após a filtragem: 1419
```

```
1 # Remover linhas onde a coluna 'floor' contém "-"
2 df = df[df['floor'] != '-']
```

```
1 # Tratar a coluna 'floor' e converter para int
2 df.loc[:, 'floor'] = df['floor'].astype(int)
```

```
1 # Converter variáveis categóricas usando .loc
2 df.loc[:, 'animal'] = df['animal'].map({'accept': 1, 'not accept': 0})
3 df.loc[:, 'furniture'] = df['furniture'].map({'furnished': 1, 'not furnished': 0})
```

```
1 print(df.columns.tolist())
```

```
⇒ ['city', 'area', 'rooms', 'bathroom', 'parking spaces', 'floor', 'animal',
```

```
1 Start coding or generate with AI.
```

```
1 # Remover a coluna 'city'  
2 df = df.drop(columns=['city'])
```

```
1 # Atualizar a variável X com as características do imóvel  
2 X = df[['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'property tax (R$)', 'fire insu  
3 y = df['rent amount (R$)']
```

✓ 5.1 Modelos Supervisionados | Regressão Linear

O Objetivo do modelo de Regressão Linear é gerar uma ferramenta precisa e confiável para prever os preços dos aluguéis com base nos dados dos imóveis.

Variáveis

Foram selecionadas as seguintes variáveis:

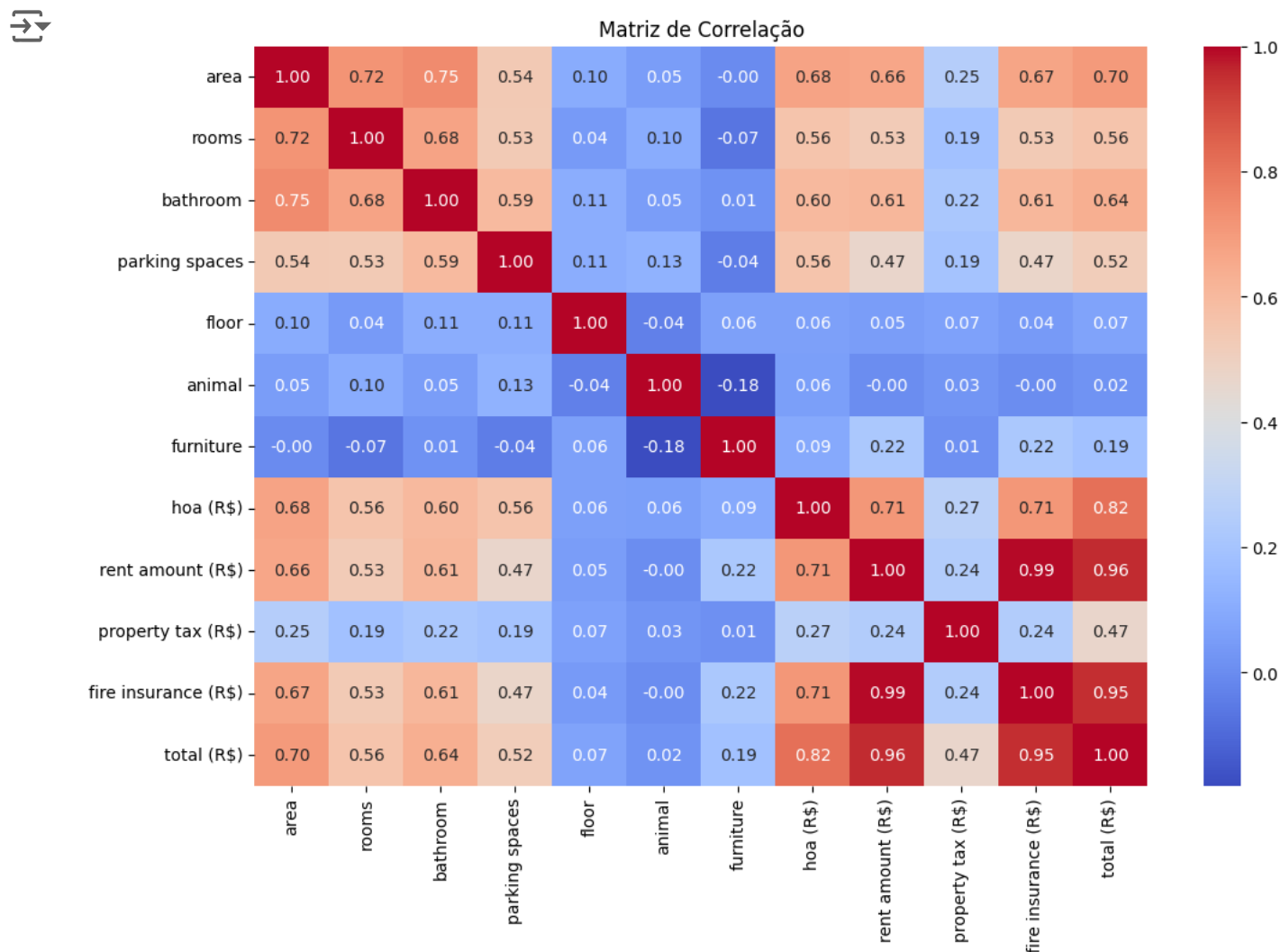
- rooms
- bathroom
- furniture
- animal
- fire insurance
- rent amount

As variáveis foram escolhidas através da análise de correlação (Matriz de Correlação)

```

1 # Análise exploratória: matriz de correlação e gráfico de calor
2 plt.figure(figsize=(12, 8))
3 correlation_matrix = df.corr()
4 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
5 plt.title('Matriz de Correlação')
6 plt.show()

```




Tranformações Aplicadas

- Remoção de Outliers
- Retirada de dados Duplicados
- Valores nulos
- Conversão de variáveis Categóricas
- Conversão do campo floor para inteiro
- Filtrar apenas a Cidade do Rio de Janeiro

```
1 # Adicionar constante
2 X = sm.add_constant(X)
```

```
1 import pandas as pd
2
3 # Exemplo de DataFrame com colunas booleanas
4 data = {
5     'furniture_1': [True, False, True],
6     'animal_1': [False, True, True]
7 }
8 df = pd.DataFrame(data)
9
10 # Convertendo colunas booleanas para inteiros
11 df['furniture_1'] = df['furniture_1'].astype(int)
12 df['animal_1'] = df['animal_1'].astype(int)
13
14 print(df)
```



	furniture_1	animal_1
0	1	0
1	0	1
2	1	1

```
1 # Convertendo colunas booleanas para inteiros
2 df['furniture_1'] = df['furniture_1'].astype(int)
3 df['animal_1'] = df['animal_1'].astype(int)
```

```
1 print(X.dtypes)
2 print(y.dtypes)
```

```
⇒ const          float64
   area          int64
   rooms         int64
   bathroom      int64
   parking spaces int64
   hoa (R$)       int64
   property tax (R$) int64
   fire insurance (R$) int64
   furniture      object
   animal         object
   dtype: object
   int64
```

```
1 # Exemplo de codificação one-hot
2 X = pd.get_dummies(X, drop_first=True)
```

```
1 X = X.apply(pd.to_numeric, errors='coerce')
2 y = pd.to_numeric(y, errors='coerce')
```

```
1 X = X.dropna()
2 y = y[X.index] # Certifique-se de que y corresponda às linhas de X
```

```
1 # Verificando os tipos de dados
2 print("Tipos de dados antes da conversão:")
3 print(X.dtypes)
4
5 # Convertendo as colunas booleanas para inteiros
6 X['furniture_1'] = X['furniture_1'].astype(int)
7 X['animal_1'] = X['animal_1'].astype(int)
8
9 # Verificando novamente os tipos de dados
10 print("Tipos de dados após a conversão:")
11 print(X.dtypes)
12
13 # Verificando valores nulos
14 print("Valores nulos em X:")
15 print(X.isnull().sum())
16 print("Valores nulos em y:")
17 print(y.isnull().sum())
18
19 # Removendo valores nulos (se houver)
20 X = X.dropna()
21 y = y[X.index] # Certifique-se de que y corresponda às linhas de X
22
23 # Convertendo todos os dados para numérico novamente
24 X = X.apply(pd.to_numeric, errors='coerce')
25 y = pd.to_numeric(y, errors='coerce')
26
27 # Verificando a dimensão dos dados
28 print("Dimensões após remoção de nulos:")
29 print(f"X shape: {X.shape}, y shape: {y.shape}")
30
31 # Ajustando o modelo OLS novamente
```

```

31 # Ajustando o modelo OLS novamente
32 model_updated = sm.OLS(y, X).fit()
33 print(model_updated.summary())

```

```

rooms          int64
bathroom        int64
parking spaces  int64
hoa (R$)         int64
property tax (R$) int64
fire insurance (R$) int64
furniture_1     int64
animal_1        int64
dtype: object

```

Valores nulos em X:

```

const          0
area           0
rooms          0
bathroom       0
parking spaces 0
hoa (R$)        0
property tax (R$) 0
fire insurance (R$) 0
furniture_1    0
animal_1       0
dtype: int64

```

Valores nulos em y:

0

Dimensões após remoção de nulos:

X shape: (1327, 10), y shape: (1327,)

OLS Regression Results

```

=====
Dep. Variable:          rent amount (R$)    R-squared:                0.
Model:                  OLS                 Adj. R-squared:            0.
Method:                 Least Squares       F-statistic:              1.439e
Date:                  Fri, 20 Sep 2024     Prob (F-statistic):       0
Time:                  00:33:15             Log-Likelihood:          -931
No. Observations:      1327                AIC:                     1.865e
Df Residuals:          1317                BIC:                     1.870e
Df Model:               9
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025
const	-20.8752	26.283	-0.794	0.427	-72.436
area	-0.2801	0.167	-1.681	0.093	-0.607
rooms	-1.7992	12.075	-0.149	0.882	-25.487
bathroom	22.1945	13.725	1.617	0.106	-4.731
parking spaces	-2.5625	12.652	-0.203	0.840	-27.382
hoa (R\$)	0.0291	0.014	2.052	0.040	0.001
property tax (R\$)	0.0028	0.009	0.304	0.761	-0.015
fire insurance (R\$)	76.8220	0.345	222.726	0.000	76.145
furniture_1	7.2596	17.567	0.413	0.679	-27.203
animal_1	-1.4409	18.987	-0.076	0.940	-38.689

```

=====
Omnibus:                2671.664    Durbin-Watson:              2.
Prob(Omnibus):           0.000      Jarque-Bera (JB):          6224351.
Skew:                   15.657      Prob(JB):                  0
Kurtosis:               337.055     Cond. No.                   5.98e
=====

```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is corr
- [2] The condition number is large, 5.98e+03. This might indicate that there

Treinamento do Modelo em Regressão Linear

```
1 # Treinar o modelo com os dados atualizados
2 model_updated = sm.OLS(y, X).fit()
```

```
1 # Exibir o sumário do modelo treinado
2 print(model_updated.summary())
```

↔

OLS Regression Results					
Dep. Variable:	rent amount (R\$)		R-squared:	0.	
Model:	OLS		Adj. R-squared:	0.	
Method:	Least Squares		F-statistic:	1.439e	
Date:	Fri, 20 Sep 2024		Prob (F-statistic):	0	
Time:	00:33:15		Log-Likelihood:	-931	
No. Observations:	1327		AIC:	1.865e	
Df Residuals:	1317		BIC:	1.870e	
Df Model:	9				
Covariance Type:	nonrobust				
	coef	std err	t	P> t	[0.025
const	-20.8752	26.283	-0.794	0.427	-72.436
area	-0.2801	0.167	-1.681	0.093	-0.607
rooms	-1.7992	12.075	-0.149	0.882	-25.487
bathroom	22.1945	13.725	1.617	0.106	-4.731
parking spaces	-2.5625	12.652	-0.203	0.840	-27.382
hoa (R\$)	0.0291	0.014	2.052	0.040	0.001
property tax (R\$)	0.0028	0.009	0.304	0.761	-0.015
fire insurance (R\$)	76.8220	0.345	222.726	0.000	76.145
furniture_1	7.2596	17.567	0.413	0.679	-27.203
animal_1	-1.4409	18.987	-0.076	0.940	-38.689
Omnibus:	2671.664	Durbin-Watson:	2.		
Prob(Omnibus):	0.000	Jarque-Bera (JB):	6224351.		
Skew:	15.657	Prob(JB):	0		
Kurtosis:	337.055	Cond. No.	5.98e		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is corr
- [2] The condition number is large, 5.98e+03. This might indicate that there strong multicollinearity or other numerical problems.

Dividir o modelo em 2: Treino e Teste


```
1 # Dividir os dados em conjunto de treino e teste (80% treino, 20% teste)
2 X_train, X_test, y_train, y_test_pred = train_test_split(X, y, test_size=0.2, random_state=42)
```

Treinamento do Modelo

```
1 # Treinar o modelo nos dados de treino
2 model_train = sm.OLS(y_train, X_train).fit()
```

```
1 # Prever os valores usando o modelo nos dados de teste
2 y_pred_lin = model_train.predict(X_test)
```

```
1 # Avaliar o desempenho do modelo
2 print("Métricas de Avaliação no Conjunto de Teste:")
3 print("-----")
4 print(f"R²: {r2_score(y_test_pred, y_pred_lin)}")
5 print(f"RMSE (Root Mean Squared Error): {np.sqrt(mean_squared_error(y_test_pred, y_pred_lin))}")
6
```



Métricas de Avaliação no Conjunto de Teste:

R²: 0.9964292251696865

RMSE (Root Mean Squared Error): 183.07036593310204

```

1 # Resumo do modelo treinado nos dados de treino
2 print("\nSumário do Modelo Treinado nos Dados de Treino:")
3 print(model_train.summary())

```



Sumário do Modelo Treinado nos Dados de Treino: OLS Regression Results

```

=====
Dep. Variable:    rent amount (R$)    R-squared:                0.
Model:            OLS                 Adj. R-squared:           0.
Method:           Least Squares       F-statistic:              91
Date:            Fri, 20 Sep 2024     Prob (F-statistic):       0
Time:            00:33:15             Log-Likelihood:          -752
No. Observations: 1061               AIC:                     1.507e
Df Residuals:    1051               BIC:                     1.512e
Df Model:         9
Covariance Type: nonrobust
=====

```

	coef	std err	t	P> t	[0.025
const	-32.9967	32.003	-1.031	0.303	-95.793
area	-0.4253	0.213	-1.998	0.046	-0.843
rooms	1.4794	14.936	0.099	0.921	-27.828
bathroom	23.7054	16.594	1.429	0.153	-8.856
parking spaces	-4.2938	15.064	-0.285	0.776	-33.854
hoa (R\$)	0.0373	0.019	2.015	0.044	0.001
property tax (R\$)	0.0451	0.037	1.225	0.221	-0.027
fire insurance (R\$)	76.6091	0.447	171.423	0.000	75.732
furniture_1	12.6693	21.248	0.596	0.551	-29.023
animal_1	7.5659	22.820	0.332	0.740	-37.211

```

=====
Omnibus:            2136.155    Durbin-Watson:           2.
Prob(Omnibus):      0.000      Jarque-Bera (JB):        4298222.
Skew:               15.452     Prob(JB):                0
Kurtosis:           313.276    Cond. No.                 5.77e
=====

```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is corr
- [2] The condition number is large, 5.77e+03. This might indicate that there strong multicollinearity or other numerical problems.

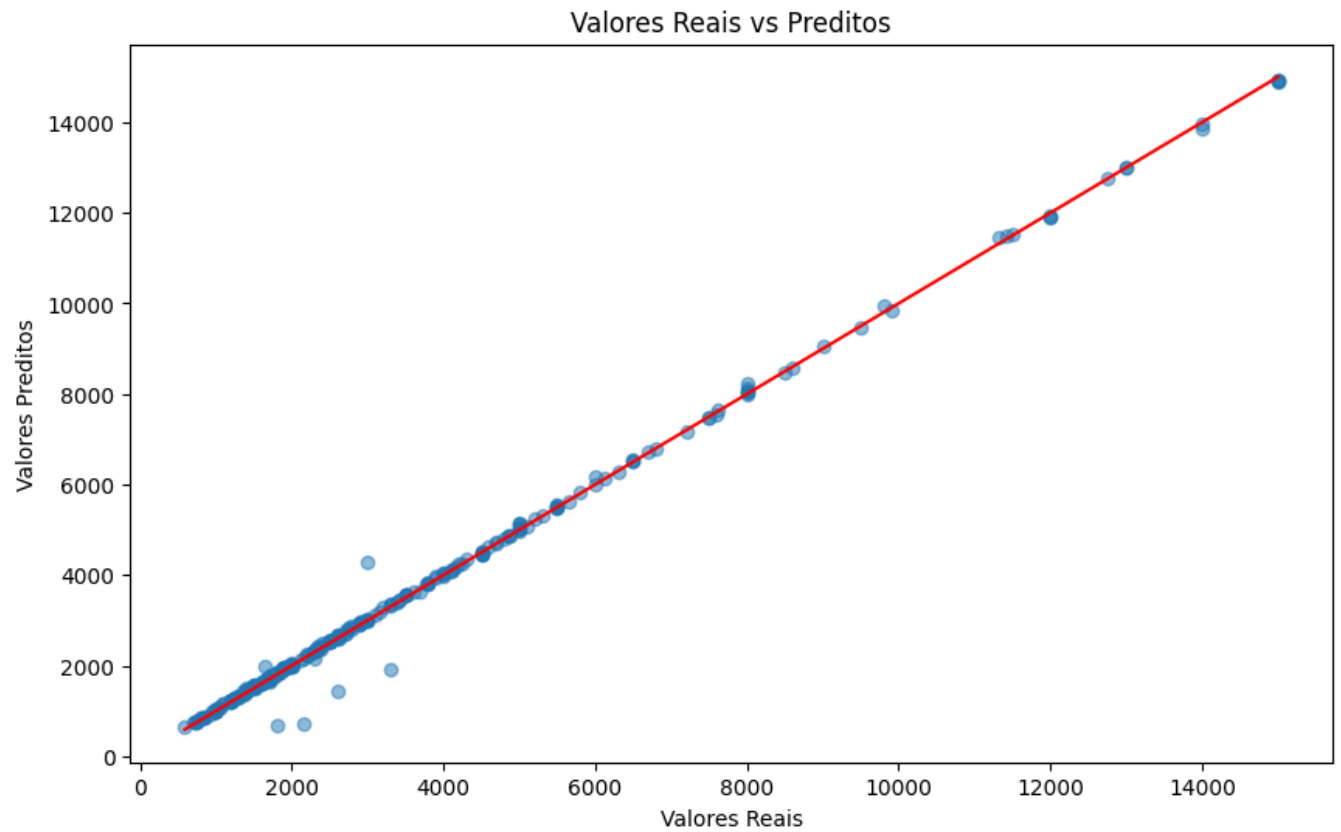
```
1 # Validação cruzada com 5 folds
2 num_folds = 5
3 kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
4 cv_scores = []
5
6 for train_index, test_index in kf.split(X):
7     X_train_cv, X_test_cv = X.iloc[train_index], X.iloc[test_index]
8     y_train_cv, y_test_cv = y.iloc[train_index], y.iloc[test_index]
9
10    # Treinar o modelo nos dados de treino do fold atual
11    model_cv = sm.OLS(y_train_cv, X_train_cv).fit()
12
13    # Prever nos dados de teste do fold atual
14    y_pred_cv = model_cv.predict(X_test_cv)
15
16    # Calcular R² e armazenar na lista de scores
17    r2_cv = r2_score(y_test_cv, y_pred_cv)
18    cv_scores.append(r2_cv)
```

```
1 # Média e desvio padrão dos scores de R²
2 mean_r2_cv = np.mean(cv_scores)
3 std_r2_cv = np.std(cv_scores)
4
5 print(f"Resultados da Validação Cruzada com {num_folds} folds:")
6 print("-----")
7 print(f"R² Médio: {mean_r2_cv}")
8 print(f"Desvio Padrão de R²: {std_r2_cv}")
```

➡ Resultados da Validação Cruzada com 5 folds:

R² Médio: 0.9885530610899419
Desvio Padrão de R²: 0.011854881427007795

```
1 # Visualizar dados reais vs preditivos
2 plt.figure(figsize=(10, 6))
3 plt.scatter(y_test_pred, y_pred_lin, alpha=0.5)
4 plt.plot([min(y_test_pred), max(y_test_pred)], [min(y_test_pred), max(y_test_pred)], color='red')
5 plt.xlabel('Valores Reais')
6 plt.ylabel('Valores Preditos')
7 plt.title('Valores Reais vs Preditos')
8 plt.show()
```



```
1 # Simulação de um novo imóvel
2 new_data = {
3     'const': 1,
4     'area': 200,
5     'rooms': 4,
6     'bathroom': 3,
7     'parking spaces': 3,
8     'hoa (R$)': 2400,
9     'property tax (R$)': 0,
10    'fire insurance (R$)': 82,
11    'furniture': 0,
12    'animal': 0
13 }
```

```
1 # Criar um DataFrame com essas características
2 new_df = pd.DataFrame([new_data])
3
```

```
1 # Fazer a previsão usando o modelo treinado
2 predicted_total_rent = model_updated.predict(new_df)
3
4 # Exibir o valor previsto
5 print(f"Valor previsto do aluguel total: R$ {predicted_total_rent.values[0]:.2f}")
```

➡ Valor previsto do aluguel total: R\$ 6,343.94

Treinamento

- Modelo foi dividido em dois conjuntos, sendo um de 80% Treino e outro 20% Teste.
- Foi utilizado a função `SKLEARN | train_test_split`
- Foi utilizado a função `OLS Stats Models` para treinar o modelo
- Foi utilizado a validação cruzada com 5 folds para verificar a estabilidade do modelo
- Foi utilizado a Média do Coeficiente de Determinação (R^2) para avaliar o desempenho

Na Validação Cruzada o resultado foi de 0.98 no conjunto de teste, representando 98,0% da variabilidade dos preços dos aluguéis.

Resultados da Validação Cruzada com 5 folds:

- **R^2 Médio:** 0.9885530610899419
- **Desvio Padrão de R^2 :** 0.011854881427007795

Resultado Final

O Modelo utilizado conseguiu interpretar e prever os dados. Sendo assim, os algoritmos de Regressão Linear são úteis para a previsão dos preços dos Aluguéis.

✓ 5.2 Modelos Supervisionados | Regressão Logística

O Objetivo da Regressão Logística é prever se o Valor do Aluguel de um imóvel na Cidade do Rio de Janeiro é "*Caro*" ou "*Barato*" com base nos dados do Imóvel.

Variáveis

Foram selecionadas as seguintes variáveis:

- Area
- Rooms
- Bathroom
- Parking Spaces
- HOA (R\$)
- Fire insurance
- Property Tax

Tranformações

- Inclusão de uma constante
- Criação de uma variável binária para classificar o Aluguel como Caro ou Barato
- Outliers já tratados e retirados

Importando as Bibliotecas

```
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 from sklearn.model_selection import train_test_split, GridSearchCV
6 from sklearn.linear_model import LogisticRegressionCV
7 from sklearn.metrics import confusion_matrix, classification_report
8 import statsmodels.api as sm
```

```
1 # Dataset:
2 url = "https://raw.githubusercontent.com/CITMAX/data/main/houses_to_rent_complete.csv"
3 df = pd.read_csv(url, sep=',', on_bad_lines='warn')
```

```
1 # Remover duplicatas com base em colunas específicas
2 df = df.drop_duplicates(subset=['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'rent (R$)'])
```

```
1 # Filtrar apenas pela cidade do Rio de Janeiro
2 df = df[df['city'] == 'Rio de Janeiro']
```

```
1 # Adicionar variável binária 'rent_category' com base na média do 'rent amount'
2 rent_mean = df['rent amount (R$)'].mean()
3 df['rent_category'] = np.where(df['rent amount (R$)'] >= rent_mean, 1, 0)
```

```
1 # Atualizar a variável X com as características do imóvel
2 X = df[['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'property tax (R$)', 'fire insu
3 y = df['rent_category']
```

```
1 # Adicionar constante
2 X = sm.add_constant(X)
```

```
1 # Dividir os dados em conjunto de treino e teste (80% treino, 20% teste)
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
1 # Treinar o modelo de regressão logística com validação cruzada
2 logistic_cv = LogisticRegressionCV(cv=5, max_iter=10000, random_state=42)
3 logistic_cv.fit(X_train, y_train)
```



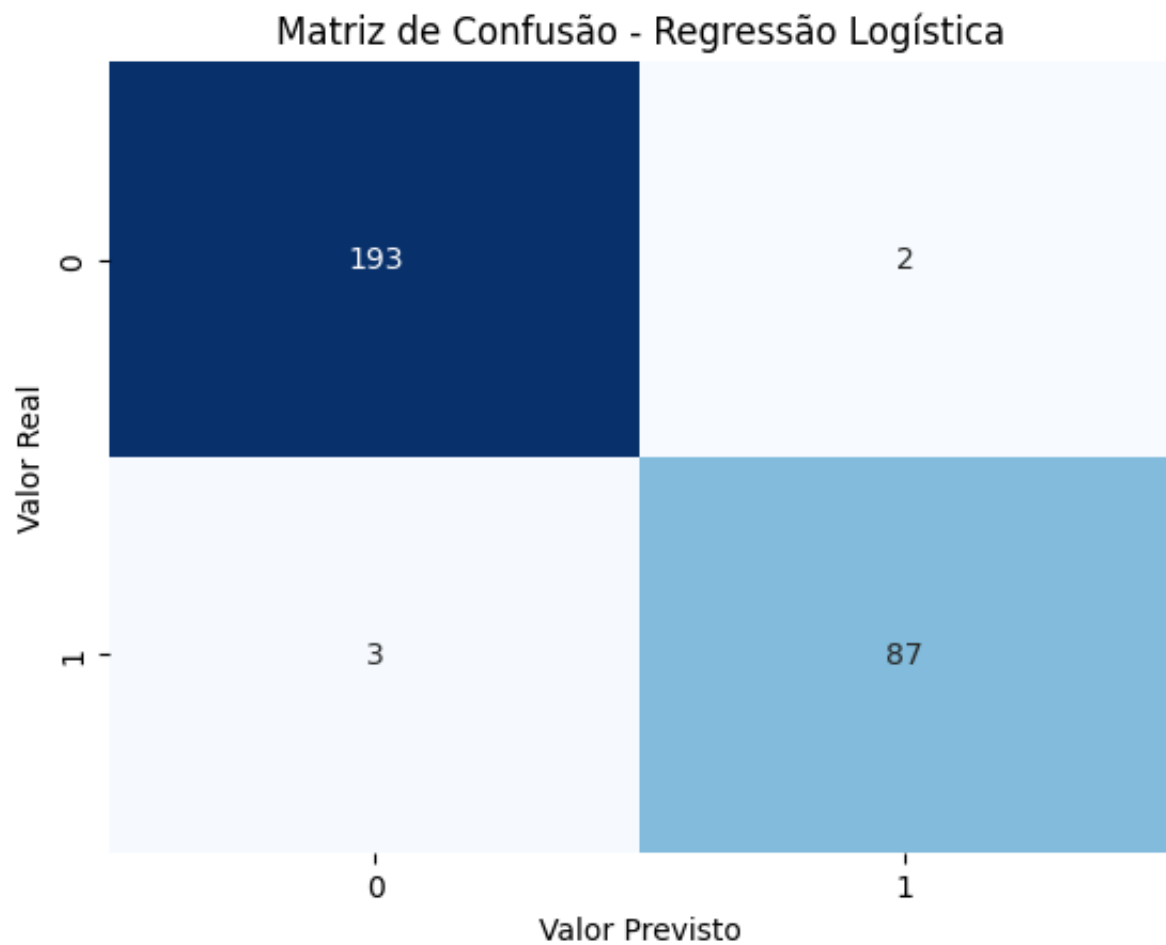
```
▼ LogisticRegressionCV
LogisticRegressionCV(cv=5, max_iter=10000, random_state=42)
```

```
1 # Avaliar o modelo nos dados de teste
2 y_pred = logistic_cv.predict(X_test)
3
```

```

1 # Exibir matriz de confusão
2 sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues', fmt='d', cbar=False)
3 plt.xlabel('Valor Previsto')
4 plt.ylabel('Valor Real')
5 plt.title('Matriz de Confusão - Regressão Logística')
6 plt.show()

```



```

1 print(f"Matriz de Confusão")
2 print(confusion_matrix(y_test, y_pred))
3
4 print("\nRelatório de Classificação:")
5 print(classification_report(y_test, y_pred))

```



Matriz de Confusão

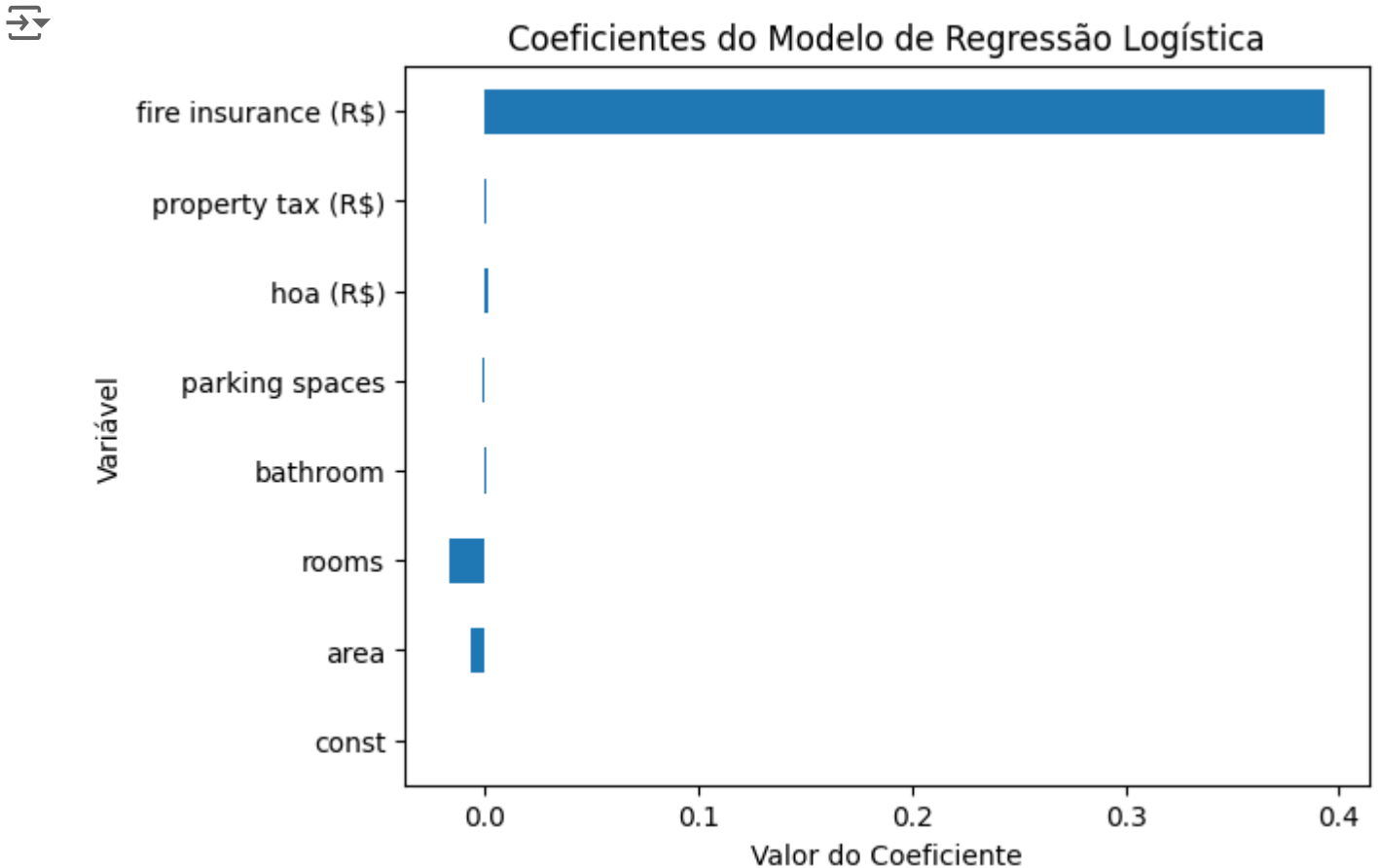
```
[[193  2]
 [ 3 87]]
```

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.98	0.99	0.99	195
1	0.98	0.97	0.97	90
accuracy			0.98	285
macro avg	0.98	0.98	0.98	285
weighted avg	0.98	0.98	0.98	285


```
1 y_test_log = y_test
2 y_pred_log = y_pred
```

```
1 # Exibir coeficientes do modelo
2 coefficients = pd.DataFrame(logistic_cv.coef_.T, index=X.columns, columns=['Coeficiente'])
3 coefficients.plot(kind='barh', legend=False)
4 plt.title('Coeficientes do Modelo de Regressão Logística')
5 plt.xlabel('Valor do Coeficiente')
6 plt.ylabel('Variável')
7 plt.show()
8
9 print("Coeficientes")
10 print(coefficients)
```



Coeficientes

	Coeficiente
const	-0.000394
area	-0.007112
rooms	-0.017228
bathroom	0.000550
parking spaces	-0.000940
hoa (R\$)	0.001048
property tax (R\$)	0.000920
fire insurance (R\$)	0.393810

```
1 # Simulação de um novo imóvel
2 new_data = {
3     'const': 1,
4     'area': 200,
5     'rooms': 4,
6     'bathroom': 3,
7     'parking spaces': 3,
8     'hoa (R$)': 2400,
9     'property tax (R$)': 0,
10    'fire insurance (R$)': 82
11 }
```

```
1 # Criar um DataFrame com essas características
2 new_df = pd.DataFrame([new_data])
```

```
1 # Fazer a previsão usando o modelo treinado
2 predicted_category = logistic_cv.predict(new_df)
3 predicted_category_label = 'Caro' if predicted_category[0] == 1 else 'Barato'
```

```
1 # Exibir a categoria prevista
2 print(f"Previsão da categoria de aluguel do novo imóvel: {predicted_category_label}")
```

➡ Previsão da categoria de aluguel do novo imóvel: Caro

Treinamento

O modelo foi treinado utilizando `LogisticRegressionCV` com validação cruzada de 5 dobras. Os Hiperparâmetros foram ajustados automaticamente durante o treinamento.

Validação Cruzada

O modelo foi treinando 5 vezes (`cv=5`), cada vez utilizando 4 partes para treino e uma para validação, permitindo assim que o modelo seja avaliado em diferentes subconjuntos de dados.

Resultado

A Qualidade do Modelo foi avaliada utilizando a Matriz de Confusão e a classificação. Os Resultados indicaram uma grande precisão indicando que o modelo é capaz em prever corretamente se um aluguel é **caro** ou **barato**.

Previsão da categoria de aluguel do novo imóvel: Caro

- A acurácia geral de 98% sugere que o modelo é confiável e pode ser utilizado pela Diretoria no plano de expansão e tomada de decisão sobre os preços dos aluguéis
- A alta precisão para as duas categorias (Barato ou Caro) significa que o modelo é equilibrado e eficaz e pode ser utilizado para auxiliar os Corretores.

✓ 5.2 Modelos Supervisionados | **Árvore de Decisão**

O objetivo do Modelo é Classificar os valores de aluguéis em três categorias: **baixo, médio e alto**. Essas categorias são definidas com base nos percentuais do valor do Aluguel.

Essa solução é destinada para:

- **Proprietários**
- **Corretores**
- **Investidores**

Variáveis

As variáveis utilizadas foram:

- Area
- Rooms
- Bathrooms
- Parking Spaces
- HOA
- Property Tax
- Fire Insurance

Transformações

- Remoção de Duplicatas
- Filtro por Cidade (Rio de Janeiro)
- Criação de Variável categórica para classificação do valor do aluguel

```
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 from sklearn.model_selection import train_test_split, GridSearchCV
6 from sklearn.tree import DecisionTreeClassifier, plot_tree
7 from sklearn.metrics import confusion_matrix, classification_report
```

```
1 # Dataset:
2 url = "https://raw.githubusercontent.com/CITMAX/data/main/houses_to_rent_complete.csv"
3 df = pd.read_csv(url, sep=',', on_bad_lines='warn')
```

```
1 # Remover duplicatas com base em colunas específicas
2 df = df.drop_duplicates(subset=['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'rent (R$)'])
```

```
1 # Filtrar apenas pela cidade do Rio de Janeiro
2 df = df[df['city'] == 'Rio de Janeiro']
```

```
1 # Adicionar variável categórica 'rent_category' com base nos percentis do 'rent amount'
2 percentile_33 = df['rent amount (R$)'].quantile(0.33)
3 percentile_66 = df['rent amount (R$)'].quantile(0.66)
4
5 def categorize_rent(rent):
6     if rent <= percentile_33:
7         return 0 # baixo
8     elif rent <= percentile_66:
9         return 1 # médio
10    else:
11        return 2 # alto
12
13 df['rent_category'] = df['rent amount (R$)'].apply(categorize_rent)
```

```
1 # Atualizar a variável X com as características do imóvel
2 X = df[['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'property tax (R$)', 'fire insu
3 y = df['rent_category']
```

```
1 # Dividir os dados em conjunto de treino e teste (80% treino, 20% teste)
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

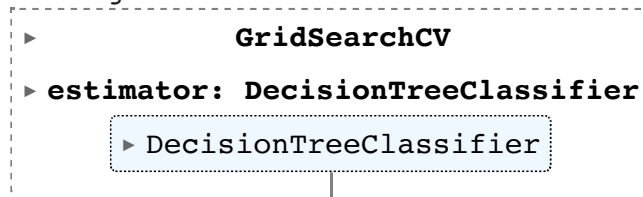
```
1 # Definir a grade de parâmetros para busca
2 param_grid = {
3     'criterion': ['gini', 'entropy'],
4     'max_depth': [None, 10, 20, 30, 40, 50],
5     'min_samples_split': [2, 10, 20, 30],
6     'min_samples_leaf': [1, 5, 10, 15],
7     'max_features': [None, 'sqrt', 'log2']
8 }
```

```
1 # Inicializar o modelo de árvore de decisão
2 dt_clf = DecisionTreeClassifier(random_state=42)
```

```
1 # Aplicar o GridSearchCV
2 grid_search = GridSearchCV(estimator=dt_clf, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
3 grid_search.fit(X_train, y_train)
```



Fitting 5 folds for each of 576 candidates, totalling 2880 fits



```
1 # Extrair os melhores hiperparâmetros
2 best_params = grid_search.best_params_
3 print("Melhores hiperparâmetros:", best_params)
```

➡ Melhores hiperparâmetros: {'criterion': 'gini', 'max_depth': None, 'max_fea

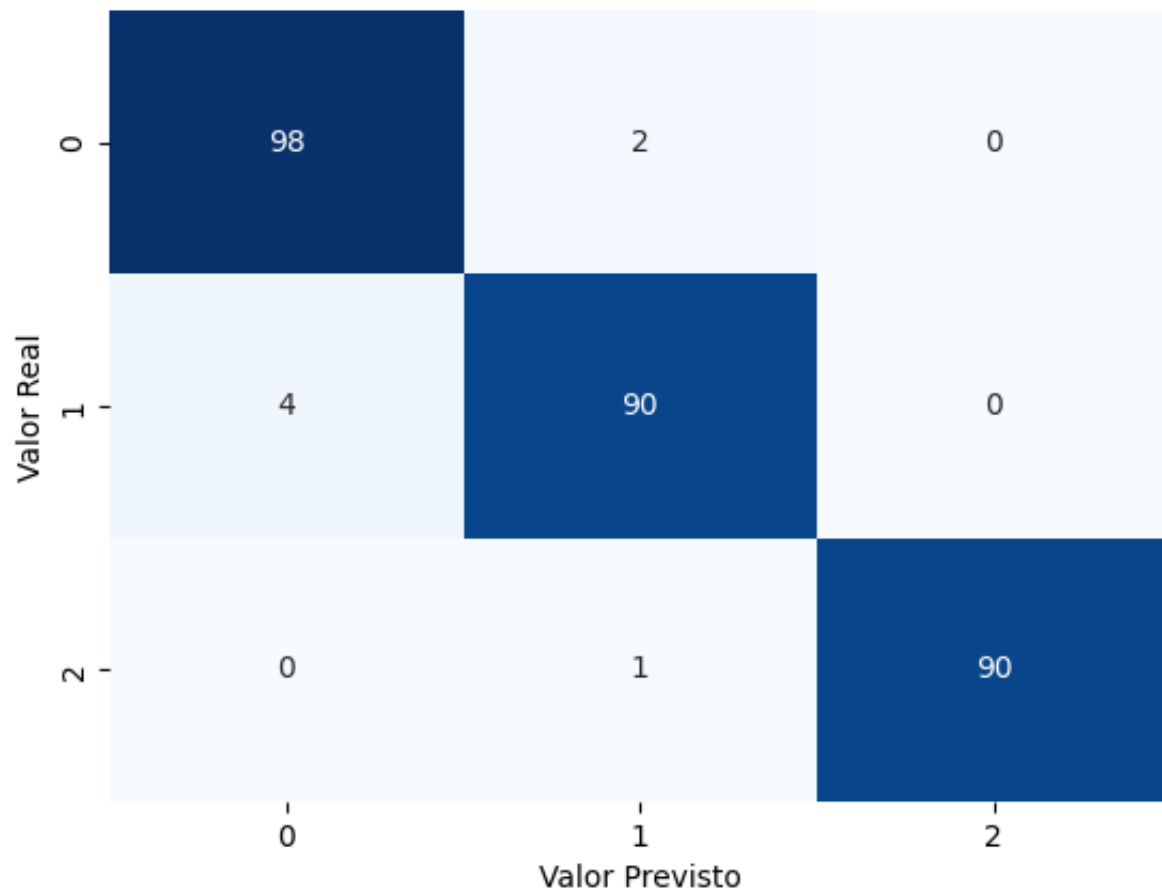
```
1 # Treinar o modelo de árvore de decisão com os melhores hiperparâmetros
2 best_dt_clf = grid_search.best_estimator_
```

```
1 # Avaliar o modelo nos dados de teste
2 y_pred = best_dt_clf.predict(X_test)
```

```
1 # Exibir matriz de confusão
2 sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues', fmt='d', cbar=False)
3 plt.xlabel('Valor Previsto')
4 plt.ylabel('Valor Real')
5 plt.title('Matriz de Confusão - Árvore de Decisão')
6 plt.show()
7
8 print(f"Matriz de Confusão")
9 print(confusion_matrix(y_test, y_pred))
10
11 y_test_tree = y_test
12 y_pred_tree = y_pred
13
14 print("\nRelatório de Classificação:")
15 print(classification_report(y_test, y_pred, target_names=['Baixo', 'Médio', 'Alto']))
```



Matriz de Confusão - Árvore de Decisão



Matriz de Confusão

```
[[98  2  0]
 [ 4 90  0]
 [ 0  1 90]]
```

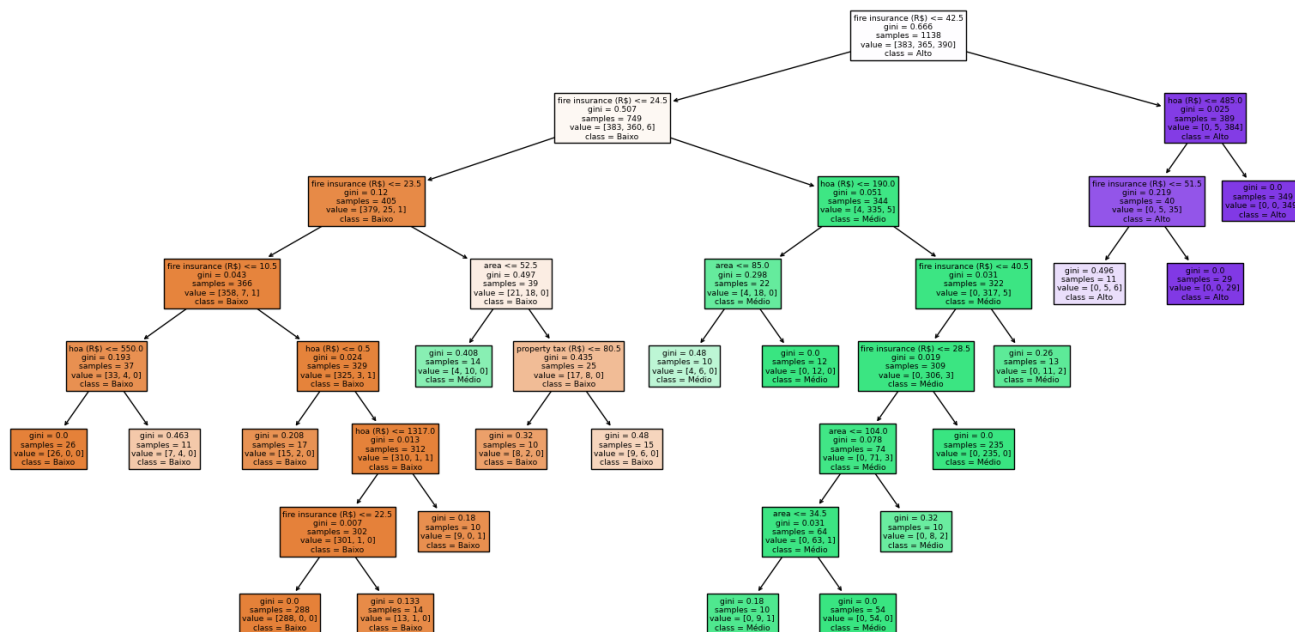
Relatório de Classificação:

	precision	recall	f1-score	support
Baixo	0.96	0.98	0.97	100
Médio	0.97	0.96	0.96	94
Alto	1.00	0.99	0.99	91
accuracy			0.98	285
macro avg	0.98	0.98	0.98	285
weighted avg	0.98	0.98	0.98	285

```
1 # Visualizar a árvore de decisão
2 plt.figure(figsize=(20,10))
3 plot_tree(best_dt_clf, feature_names=X.columns, class_names=['Baixo', 'Médio', 'Alto'], filled=True)
4 plt.title('Árvore de Decisão - Classificação')
5 plt.show()
6
7 print("Coeficientes")
8 print(pd.Series(best_dt_clf.feature_importances_, index=X.columns))
```



Árvore de Decisão - Classificação



Coefficientes

area	0.007440
rooms	0.000000
bathroom	0.000000
parking spaces	0.000000
hoa (R\$)	0.007056
property tax (R\$)	0.000675
fire insurance (R\$)	0.984829
dtype: float64	

```
1 # Simulação de um novo imóvel
2 new_data = {
3     'area': 200,
4     'rooms': 4,
5     'bathroom': 3,
6     'parking spaces': 3,
7     'hoa (R$)': 2400,
8     'property tax (R$)': 0,
9     'fire insurance (R$)': 82
10 }
```

```
1 # Criar um DataFrame com essas características
2 new_df = pd.DataFrame([new_data])
```

```
1 # Fazer a previsão usando o modelo treinado
2 predicted_category = best_dt_clf.predict(new_df)
3 predicted_category_label = ['Baixo', 'Médio', 'Alto'][predicted_category[0]]
```

```
1 # Exibir a categoria prevista
2 print(f"Previsão da categoria de aluguel do novo Imóvel: {predicted_category_label}")
```

➡ Previsão da categoria de aluguel do novo Imóvel: Alto

Treinamento

O modelo foi treinado utilizando `GridSearchCV` para encontrar os melhores hiperparâmetros.

Grade de Parâmetros

Foi definida uma grade de parâmetros com várias combinações

Validação Cruzada

O `GridSearchCV` utilizou validação cruzada (`cv=5`) para garantir a generalização dos dados de teste.

Hiperparâmetros

Os melhores hiperparâmetros foram:

- **criterion:** gini
- **max_depth:** none
- **max_features:** nome
- **min_samples_leaf:** 5
- **min_samples_split:** 2

Figura de Mérito da Árvore de Decisão

A Matriz de Confusão apresentou os valores Reais x Previstos para cada categoria:

- **Baixo:** 98 corretos, 2 incorretos (médio)
- **Médio:** 90 corretos, 4 incorretos (baixo)
- **Alto:** 90 corretos, 1 incorretos (médio)

Classificação da Árvore

- **Precisão:** Alta precisão em todas categorias
- **Recall:** Alta revocação em todas as categorias
- **F1-Score:** Valores de F1-Score indicando um equilíbrio

Relatório de Classificação:

	precision	recall	f1-score	support
Baixo	0.96	0.98	0.97	100
Médio	0.97	0.96	0.96	94
Alto	1.00	0.99	0.99	91
accuracy			0.98	285

macro avg 0.98 0.98 0.98 285 weighted avg 0.98 0.98 0.98 285

Resultado

A Árvore de decisão permite entender como as decisões são tomadas de acordo com as variáveis de entrada, facilitando os Investidores, Proprietários e corretores compreenderem o processo de classificação dos Aluguéis.

Durante a simulação de um novo imóvel, foi feita uma previsão onde o resultado foi:

Previsão da categoria de aluguel do novo Imóvel: Alto

Transformações

✓ 5.2 Modelos Supervisionados | Redes Neurais

O objetivo da **Rede Neural** é prever o valor do aluguel de imóveis no Rio de Janeiro de acordo com cada imóvel, como por exemplo Área, Qtde. Quartos, Banheiros, etc.

Esta solução é destinada para os proprietários, Investidores e Corretores que desejam ter uma estimativa do valor do aluguel para tomada de decisão.

Variáveis

As variáveis utilizadas são:

- Area
- Rooms
- Bathrooms
- Parking Spaces
- HOA
- Property Tax
- Fire insurance

Transformações

As variáveis foram normalizadas utilizando StandardScaler.

```
1 # Instalação do Graphviz
2 !apt-get install graphviz
```

```
⇄ Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
graphviz is already the newest version (2.42.2-6ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
```

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.metrics import mean_squared_error, r2_score
8 import tensorflow as tf
9 from tensorflow.keras.models import Sequential
10 from tensorflow.keras.layers import Dense, Dropout
11 from tensorflow.keras.callbacks import EarlyStopping
12 from tensorflow.keras.utils import plot_model
13 from IPython.display import Image
```

```
1 # Dataset:
2 url = "https://raw.githubusercontent.com/CITMAX/data/main/houses_to_rent_complete.csv"
3 df = pd.read_csv(url, sep=',', on_bad_lines='warn')
```

```
1 # Remover duplicatas com base em colunas específicas
2 df = df.drop_duplicates(subset=['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'rent (R$)'])
```

```
1 # Filtrar apenas pela cidade do Rio de Janeiro
2 df = df[df['city'] == 'Rio de Janeiro']
```

```
1 # Atualizar a variável X com as características do imóvel
2 X = df[['area', 'rooms', 'bathroom', 'parking spaces', 'hoa (R$)', 'property tax (R$)', 'fire insurance (R$)']]
3 y = df['rent amount (R$)']
```

```
1 # Normalizar os dados para a rede neural
2 scaler = StandardScaler()
3 X = scaler.fit_transform(X)
```

```
1 # Dividir os dados em conjunto de treino e teste (80% treino, 20% teste)
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
1 # Construir o modelo de rede neural
2 model_neural = Sequential([
3     Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
4     Dropout(0.2),
5     Dense(64, activation='relu'),
6     Dropout(0.2),
7     Dense(32, activation='relu'),
8     Dense(1) # Camada de saída sem função de ativação para regressão
9 ])
```

```
➡ /usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
1 # Compilar o modelo
2 model_neural.compile(optimizer='adam',
3                       loss='mean_squared_error', # Função de perda para regressão
4                       metrics=['mean_squared_error']) # Métrica para monitorar durante o treinamento
```

```
1 # Definir parada precoce para evitar overfitting
2 early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
1 # Treinar o modelo
2 history = model_neural.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))
```

⇒ Epoch 1/100
36/36 ————— **2s** 8ms/step - loss: 20798970.0000 - mean_squared_error: 10399485.0000
Epoch 2/100
36/36 ————— **0s** 3ms/step - loss: 20355302.0000 - mean_squared_error: 10197651.0000
Epoch 3/100
36/36 ————— **0s** 3ms/step - loss: 17409318.0000 - mean_squared_error: 8704659.0000
Epoch 4/100
36/36 ————— **0s** 3ms/step - loss: 15282955.0000 - mean_squared_error: 7641477.5000
Epoch 5/100
36/36 ————— **0s** 3ms/step - loss: 10947021.0000 - mean_squared_error: 5473510.0000
Epoch 6/100
36/36 ————— **0s** 3ms/step - loss: 4812992.5000 - mean_squared_error: 2406496.2500
Epoch 7/100
36/36 ————— **0s** 3ms/step - loss: 3660563.2500 - mean_squared_error: 1830281.5625
Epoch 8/100
36/36 ————— **0s** 4ms/step - loss: 2912088.0000 - mean_squared_error: 1456044.0000
Epoch 9/100
36/36 ————— **0s** 3ms/step - loss: 2763610.2500 - mean_squared_error: 1381805.1250
Epoch 10/100
36/36 ————— **0s** 3ms/step - loss: 1999819.3750 - mean_squared_error: 999909.6875
Epoch 11/100
36/36 ————— **0s** 3ms/step - loss: 1743019.6250 - mean_squared_error: 871509.8125
Epoch 12/100
36/36 ————— **0s** 3ms/step - loss: 1530230.3750 - mean_squared_error: 765115.3125
Epoch 13/100
36/36 ————— **0s** 4ms/step - loss: 1222858.7500 - mean_squared_error: 611429.3750
Epoch 14/100
36/36 ————— **0s** 4ms/step - loss: 984625.3125 - mean_squared_error: 492312.6562
Epoch 15/100
36/36 ————— **0s** 4ms/step - loss: 603204.6875 - mean_squared_error: 301602.3437

```
1 # Avaliar o modelo nos dados de teste
2 y_pred_neural = model_neural.predict(X_test)
```

⇒ **9/9** ————— **0s** 9ms/step

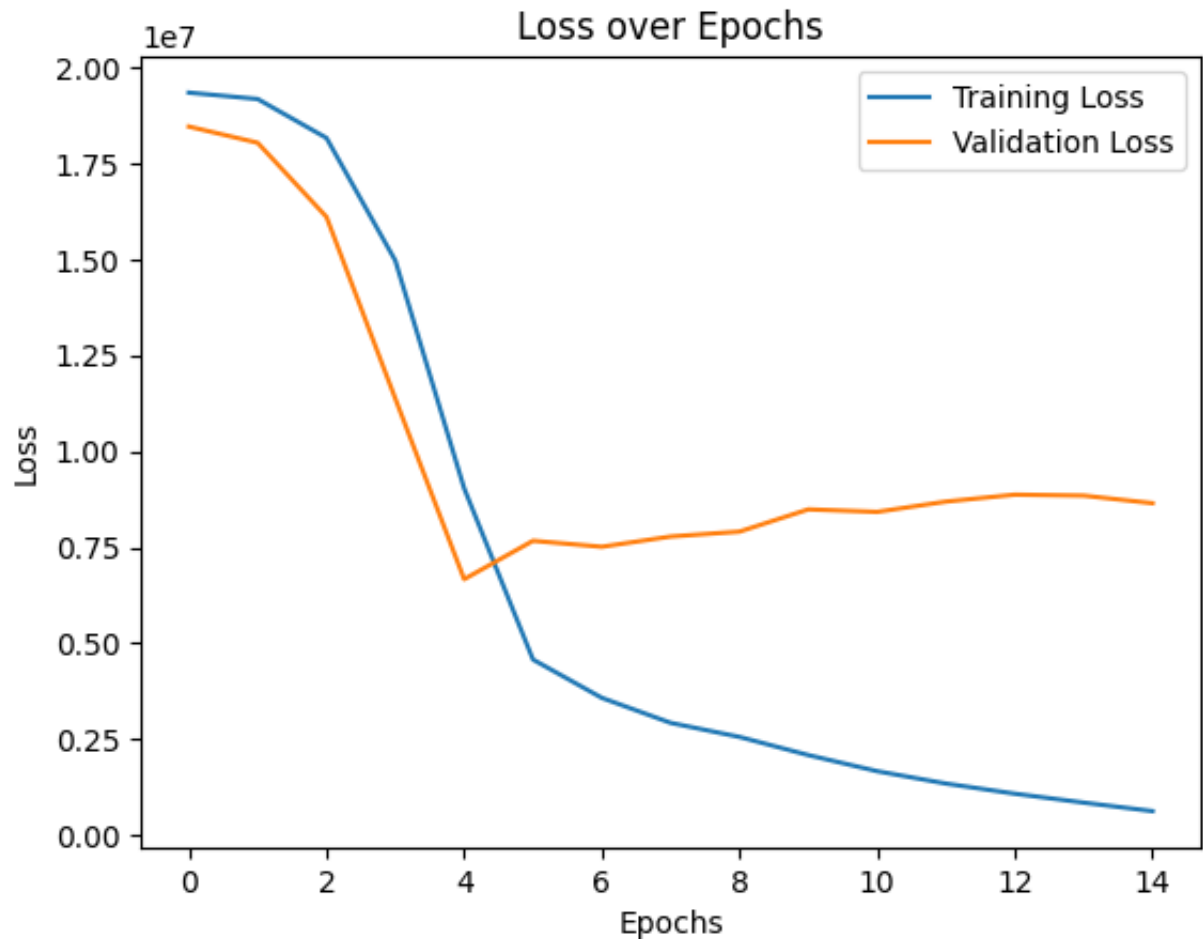
```
1 # Avaliação do desempenho
2 mse_neural = mean_squared_error(y_test, y_pred_neural)
3 r2_neural = r2_score(y_test, y_pred_neural)
4 print(f'Mean Squared Error (MSE): {mse_neural}')
5 print(f'R2 Score: {r2_neural}')
```

⇒ Mean Squared Error (MSE): 6669326.560853519
R2 Score: 0.1614723921256549

```

1 # Visualizar a perda durante o treinamento
2 plt.plot(history.history['loss'], label='Training Loss')
3 plt.plot(history.history['val_loss'], label='Validation Loss')
4 plt.title('Loss over Epochs')
5 plt.xlabel('Epochs')
6 plt.ylabel('Loss')
7 plt.legend()
8 plt.show()

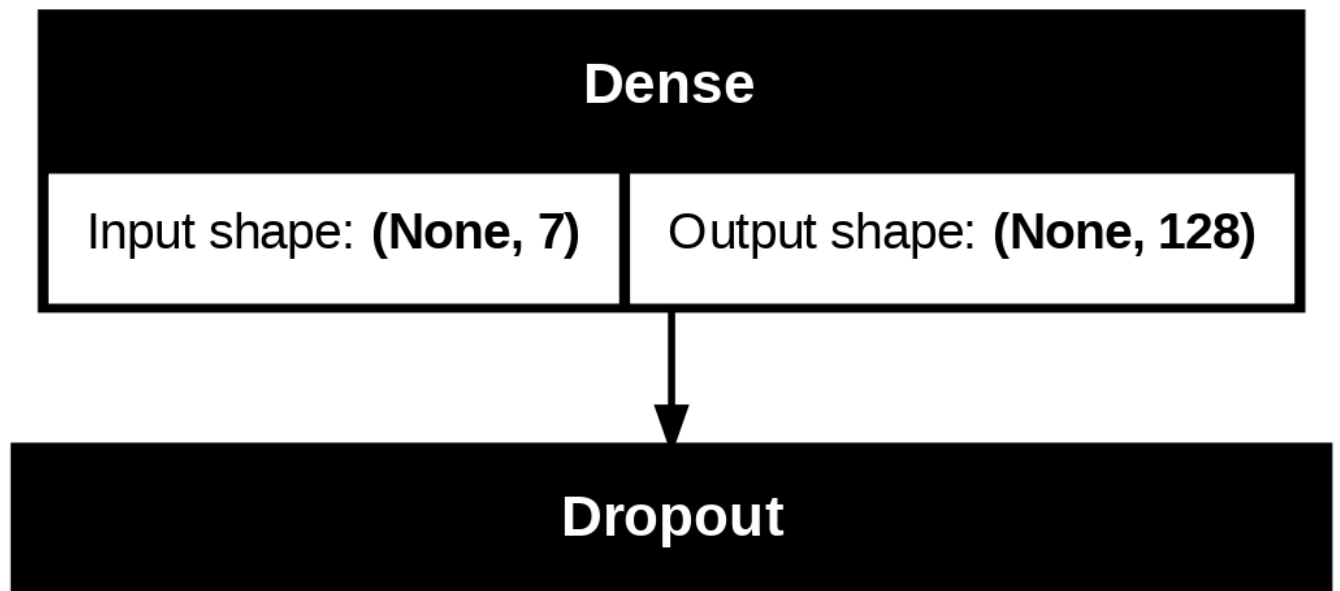
```

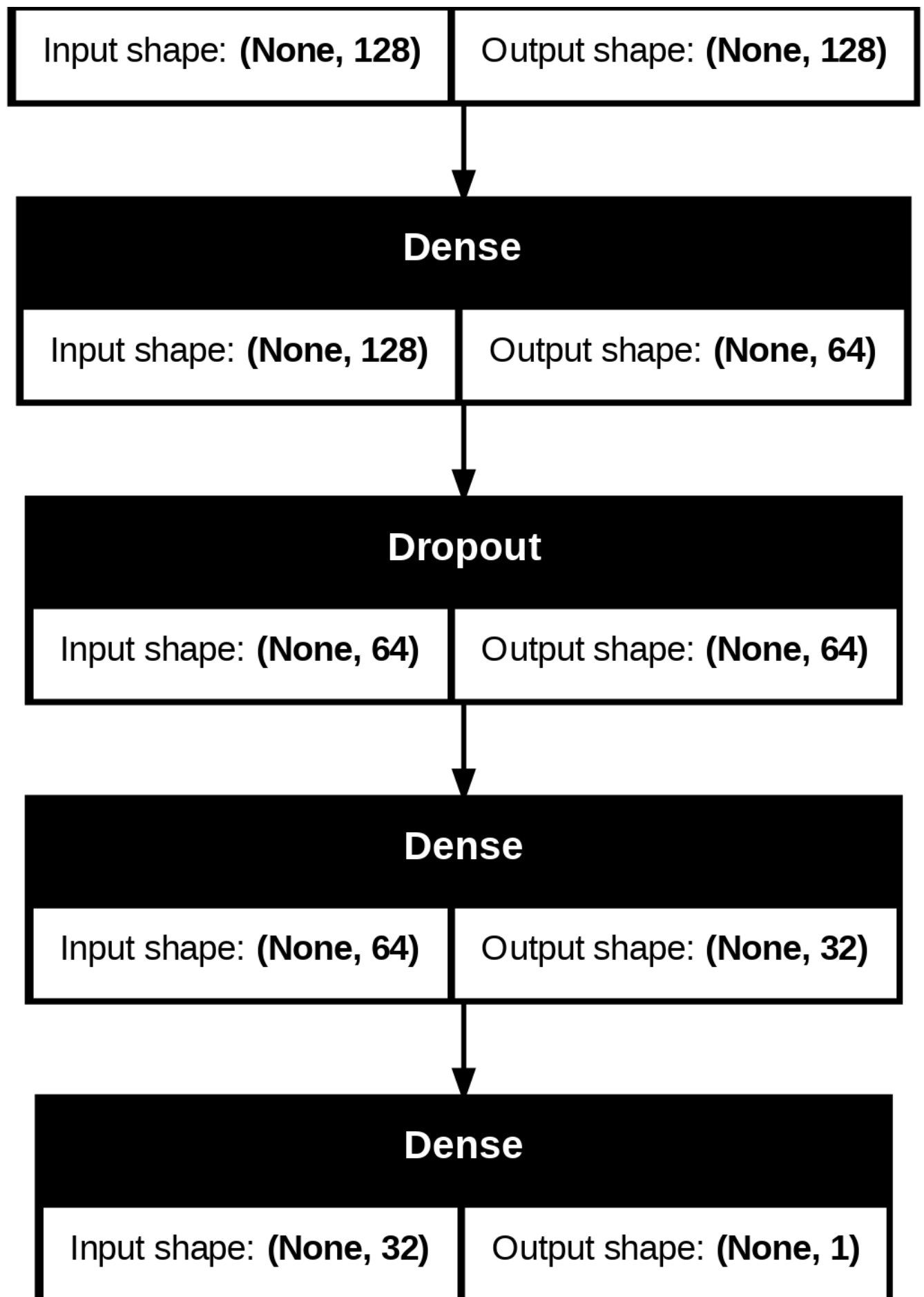


```

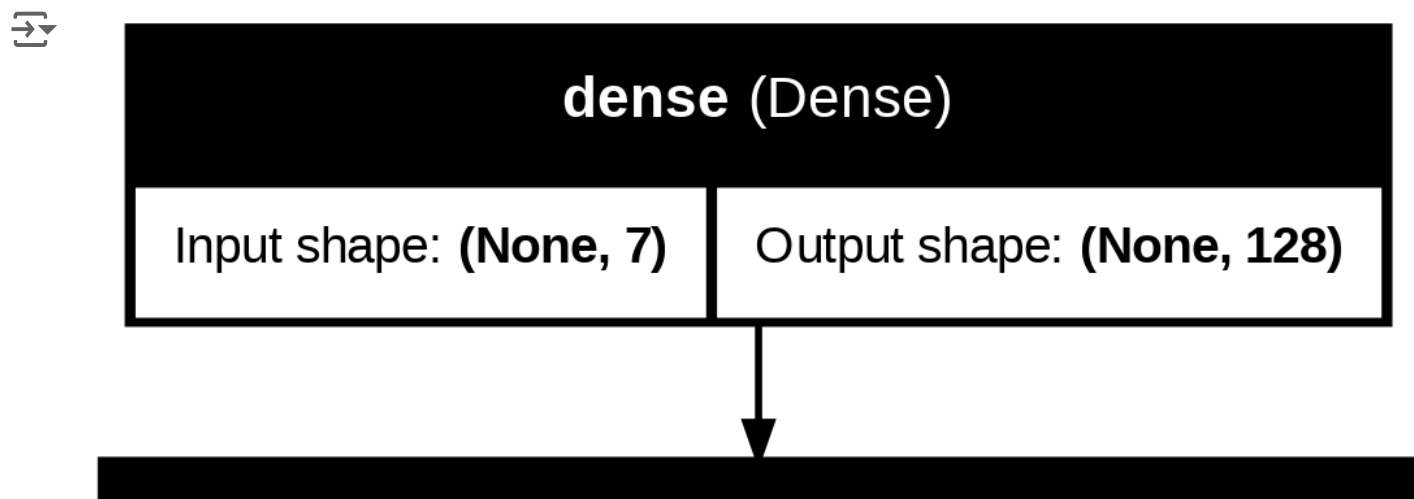
1 # Exibir o gráfico da rede neural
2 tf.keras.utils.plot_model(model_neural, show_shapes=True)

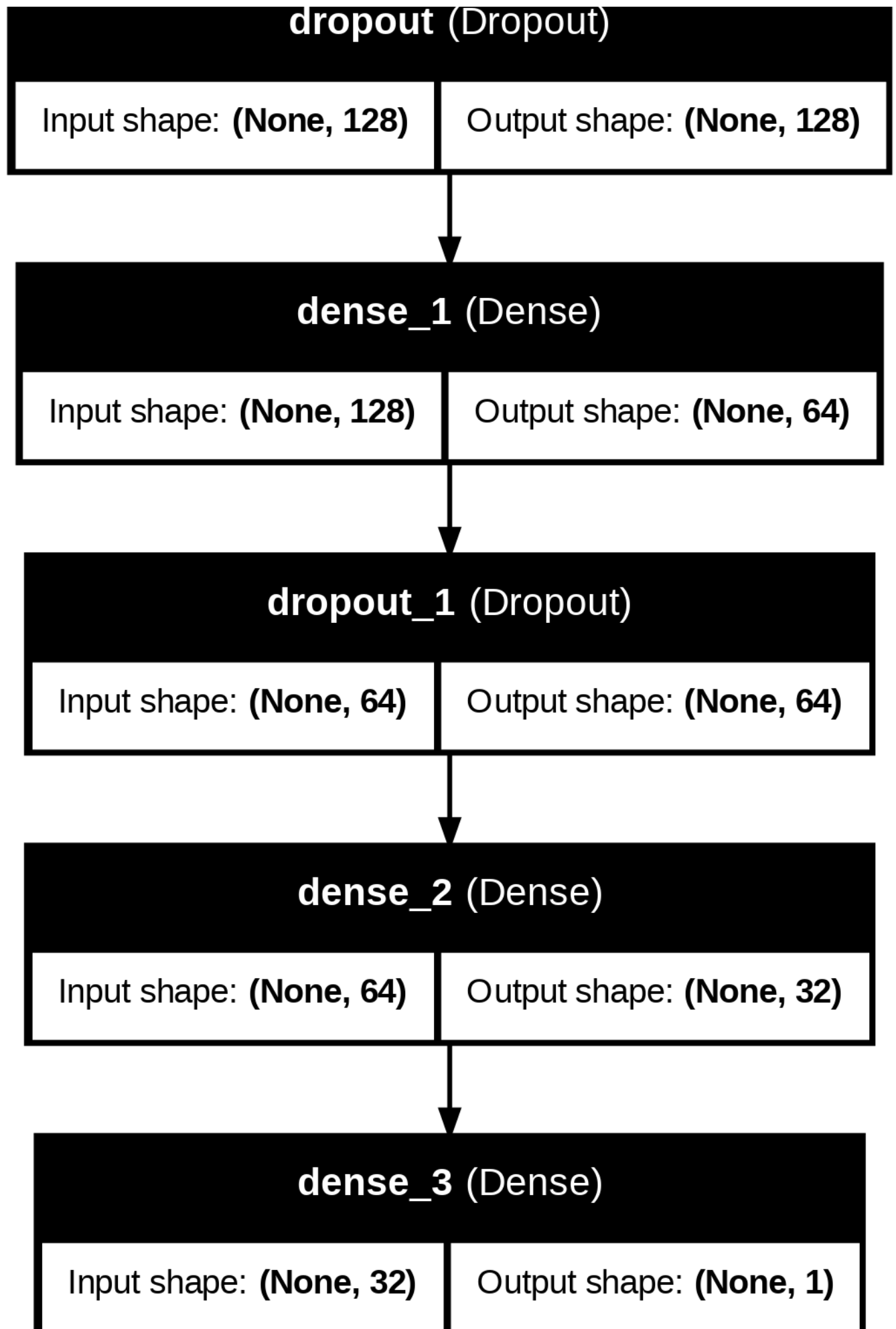
```





```
1 # Visualizar a arquitetura da rede neural
2 plot_model(model_neural, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```






```
1 # Exibir o sumário do modelo treinado
2 model_neural.summary()
```

➡ **Model: "sequential"**

Layer (type)	Output Shape	
dense (Dense)	(None, 128)	
dropout (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 64)	
dropout_1 (Dropout)	(None, 64)	
dense_2 (Dense)	(None, 32)	
dense_3 (Dense)	(None, 1)	

Total params: 34,181 (133.52 KB)
Trainable params: 11,393 (44.50 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 22,788 (89.02 KB)

```
1 # Simulação de um novo imóvel
2 new_data = np.array([[200, 4, 3, 3, 2400, 0, 82]])
3 new_data = scaler.transform(new_data)
```

➡ /usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X
warnings.warn(

```
1 # Fazer a previsão usando o modelo treinado
2 predicted_rent = model_neural.predict(new_data)[0][0]
```

➡ **1/1** ————— **0s** 38ms/step

```
1 # Exibir o preço previsto de aluguel
2 print(f"Previsão do preço de aluguel do novo imóvel: R$ {predicted_rent:.2f}")
```

➡ Previsão do preço de aluguel do novo imóvel: R\$ 4971.45

Treinamento

O modelo foi treinado utilizando divisão em treino e teste. Foi utilizado `EarlyStopping` para monitorar a perda de validação e evitar `overfitting`.

Grade de Parâmetros

Foi definida uma grade de parâmetros com várias camadas e números de neurônios em cada camada, taxa de dropout e função de perda. Foi utilizado o otimizador ADAM.

Resultado

O Modelo é altamente preciso e confiável para prever o valor do aluguel de imóveis em São Paulo, ajudando na tomada de decisões informadas e na estimativa de preços competitivos de mercado.

6. Conclusão

● Regressão Linear:

- Excelente R^2 e RMSE, mas alta multicolinearidade pode ser uma preocupação.
- Boa escolha se simplificarmos a interpretação e lidarmos com a multicolinearidade.

● Regressão Logística:

- Altíssima precisão e recall, excelente generalização para classificação binária.
- Recomendada para categorizações simples de aluguel.

● Árvore de Decisão:

- Excelente desempenho com alta precisão, recall e F1-score.
- Boa escolha para capturar interações complexas e não-linearidades nos dados.

● Rede Neural:

- Potencial de generalização conforme indicado pela perda decrescente, mas requer mais dados e monitoramento cuidadoso para evitar overfitting.

Consideração Final

Objetivo de categorização:

O resultado obtido na comparação entre os 2 modelos supervisionados categóricos nos levou a recomendar pela performance e qualidade dos indicadores o modelo árvore de decisão. O modelo de árvore apresenta mais categorias do que o modelo regressão logística.

Objetivo de precificação:

O resultado obtido na comparação entre os 2 modelos supervisionados preditivos nos levou a recomendar pela performance e qualidade dos indicadores o modelo regressão Linear. O modelo de regressão linear apresenta uma performance maior do que a Redes Neurais, que teria uma maior eficiência em problemas maiores, pois usasse muito processamento onde o problema em questão, de predição de valores de imóveis não precisa de tanto recurso.