# MODELAGEM PREDITIVA DO VALOR DE ALUGUEIS DE APARTAMENTOS EM ALGUMAS CIDADES DO BRASIL

Kadu Vinicius Toledo Paulino

PUC MINAS

# O problema Proposto

Vamos utilizar análise exploratória e modelar o preço de aluguel de algumas cidades do Brasil através de variáveis relacionadas com o imóvel, mostrando o impacto das mesmas no preço através de Machine Learning. Temos como objetivos:

- Realizar a análise descritiva dos dados;
- Verificar a correlação entre as variáveis;
- Criar modelos preditivos para o preço dos alugueis de apartamentos de algumas cidades brasileiras;
- Buscar o melhor modelo que se encaixa em tal representação.

# Coleta de Dados

Para tratamento do problema proposto, foram utilizados dois datasets retirados do site Kaggle.
O primeiro dataset, “sao-paulo-properties-april-2019” (link1) contém cerca de 13000 apartamentos para venda e aluguel na cidade de São Paulo, Brasil. Os dados vêm de várias fontes, especialmente sites de classificados imobiliários. O conjunto de dados representa os dados anunciados no mês de abril de 2019.

# Coleta de Dados

O segundo dataset “brazilian-houses-to-rent” (link2) contém cerca de 11000 apartamentos para locação em diferentes cidades brasileiras. Os dados foram retirados de vários sites da internet através de um WebCrawler.

# Processamento/Tratamento de Dados

O processamento e o tratamento dos dados foram feitos utilizando a linguagem Python, versão 3.7.13 no ambiente Jupyter Notebook. Inicialmente foram importadas as seguintes bibliotecas que seriam utilizadas no processamento e tratamento de dados, além de análises estatísticas:

```
import pandas as pd #carregamento de arquivos e manipulação csv
import numpy as np #metodos numéricos
import seaborn as sns #visualização de gráficos
import matplotlib.pyplot as plt #visualização de gráficos
import plotly.express as px #gráficos dinâmicos
```

# Base de dados das propriedades de São Paulo

```
base_sp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13640 entries, 0 to 13639
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Price              13640 non-null  int64  
 1   Condo              13640 non-null  int64  
 2   Size               13640 non-null  int64  
 3   Rooms              13640 non-null  int64  
 4   Toilets            13640 non-null  int64  
 5   Suites             13640 non-null  int64  
 6   Parking            13640 non-null  int64  
 7   Elevator           13640 non-null  int64  
 8   Furnished          13640 non-null  int64  
 9   Swimming Pool      13640 non-null  int64  
 10  New                13640 non-null  int64  
 11  District           13640 non-null  object 
 12  Negotiation Type   13640 non-null  object 
 13  Property Type      13640 non-null  object 
 14  Latitude           13640 non-null  float64
 15  Longitude          13640 non-null  float64
dtypes: float64(2), int64(11), object(3)
memory usage: 1.7+ MB
```

# Base de dados das propriedades de São Paulo

1. Verificar se todos os tipos de dados são apartamentos;
2. Excluir algumas colunas que não serão necessárias;
3. Só trabalhar com propriedades para locação;
4. Excluir linhas que possuem tipo de negociação venda;
5. Excluir a coluna que representa o tipo de negociação;
6. Somar o valor do condomínio e o preço do aluguel, para termos o preço total;
7. Renomear as colunas;
8. Formatar a coluna preço e condomínio para float;
9. Verificar se existem outliers, algum dado inconsistente.

# Base de dados das propriedades de São Paulo

```
base_sp
```

| | Tamanho | Quartos | Banheiros | Estacionamento | Mobiliado | Condominio | Preço |
|---|---|---|---|---|---|---|---|
| **0** | 47 | 2 | 2 | 1 | 0 | 220.0 | 1150.0 |
| **1** | 45 | 2 | 2 | 1 | 0 | 148.0 | 1148.0 |
| **2** | 48 | 2 | 2 | 1 | 0 | 100.0 | 1100.0 |
| **3** | 48 | 2 | 2 | 1 | 0 | 200.0 | 1200.0 |
| **4** | 55 | 2 | 2 | 1 | 0 | 410.0 | 1710.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11205** | 73 | 1 | 2 | 1 | 0 | 595.0 | 4295.0 |
| **11206** | 208 | 4 | 4 | 3 | 1 | 3000.0 | 24000.0 |
| **11207** | 55 | 1 | 1 | 1 | 1 | 710.0 | 4510.0 |
| **11208** | 205 | 3 | 2 | 2 | 0 | 2354.0 | 7354.0 |
| **11209** | 162 | 3 | 4 | 3 | 0 | 2300.0 | 17900.0 |

7220 rows × 7 columns

# Base de dados das propriedades de algumas cidades do Brasil

```
base_fora.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10692 entries, 0 to 10691
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   city                  10692 non-null  object
 1   area                  10692 non-null  int64 
 2   rooms                 10692 non-null  int64 
 3   bathroom              10692 non-null  int64 
 4   parking spaces        10692 non-null  int64 
 5   floor                 10692 non-null  object
 6   animal                10692 non-null  object
 7   furniture             10692 non-null  object
 8   hoa (R$)              10692 non-null  int64 
 9   rent amount (R$)      10692 non-null  int64 
 10  property tax (R$)     10692 non-null  int64 
 11  fire insurance (R$)   10692 non-null  int64 
 12  total (R$)            10692 non-null  int64 
dtypes: int64(9), object(4)
memory usage: 1.1+ MB
```

# Base de dados das propriedades de algumas cidades do Brasil

1. Verificar quais cidades estão incluidas na base de dados;
2. Excluir algumas colunas que não serão necessárias;
3. Renomear as colunas;
4. Trocar não mobiliado por 0 e mobiliado por 1;
5. Formatar a coluna mobiliado para o tipo inteiro;
6. Visualizar algum dado inconsistente;
7. Verificar se existem outliers, algum dado inconsistente.

# Base de dados das propriedades de algumas cidades do Brasil

```
base_fora
```

| | Tamanho | Quartos | Banheiros | Estacionamento | Mobiliado | Condominio | Preço |
|---|---|---|---|---|---|---|---|
| **0** | 70 | 2 | 1 | 1 | 1 | 2065.0 | 5618.0 |
| **1** | 320 | 4 | 4 | 0 | 0 | 1200.0 | 7973.0 |
| **2** | 80 | 1 | 1 | 1 | 0 | 1000.0 | 3841.0 |
| **3** | 51 | 2 | 1 | 0 | 0 | 270.0 | 1421.0 |
| **4** | 25 | 1 | 1 | 0 | 0 | 0.0 | 836.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **10687** | 63 | 2 | 1 | 1 | 1 | 402.0 | 1926.0 |
| **10688** | 285 | 4 | 4 | 4 | 0 | 3100.0 | 19260.0 |
| **10689** | 70 | 3 | 3 | 0 | 1 | 980.0 | 7390.0 |
| **10690** | 120 | 2 | 2 | 2 | 1 | 1585.0 | 14020.0 |
| **10691** | 80 | 2 | 1 | 0 | 0 | 0.0 | 1587.0 |

10664 rows × 7 columns

# Juntando as duas bases de Dados

```
base_total
```

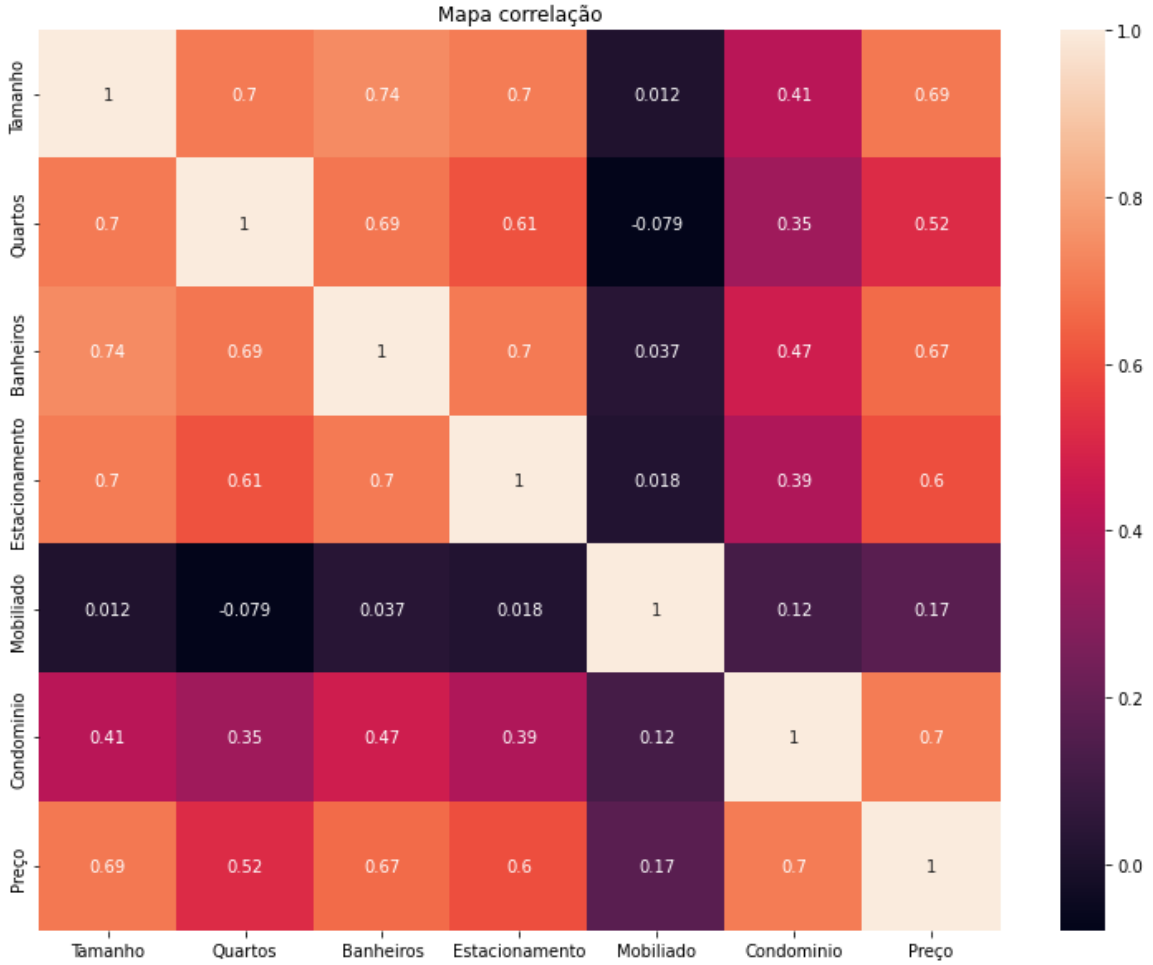
| | Tamanho | Quartos | Banheiros | Estacionamento | Mobiliado | Condominio | Preço |
|---|---|---|---|---|---|---|---|
| **0** | 47 | 2 | 2 | 1 | 0 | 220.0 | 1150.0 |
| **1** | 45 | 2 | 2 | 1 | 0 | 148.0 | 1148.0 |
| **2** | 48 | 2 | 2 | 1 | 0 | 100.0 | 1100.0 |
| **3** | 48 | 2 | 2 | 1 | 0 | 200.0 | 1200.0 |
| **4** | 55 | 2 | 2 | 1 | 0 | 410.0 | 1710.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **10687** | 63 | 2 | 1 | 1 | 1 | 402.0 | 1926.0 |
| **10688** | 285 | 4 | 4 | 4 | 0 | 3100.0 | 19260.0 |
| **10689** | 70 | 3 | 3 | 0 | 1 | 980.0 | 7390.0 |
| **10690** | 120 | 2 | 2 | 2 | 1 | 1585.0 | 14020.0 |
| **10691** | 80 | 2 | 1 | 0 | 0 | 0.0 | 1587.0 |

17884 rows × 7 columns

# Análise e Exploração dos Dados

1. Analise estatitica do df;
2. Histograma das variáveis;
3. Correlação entre as variáveis;
4. Visualização dos dados.

# Análise e Exploração dos Dados



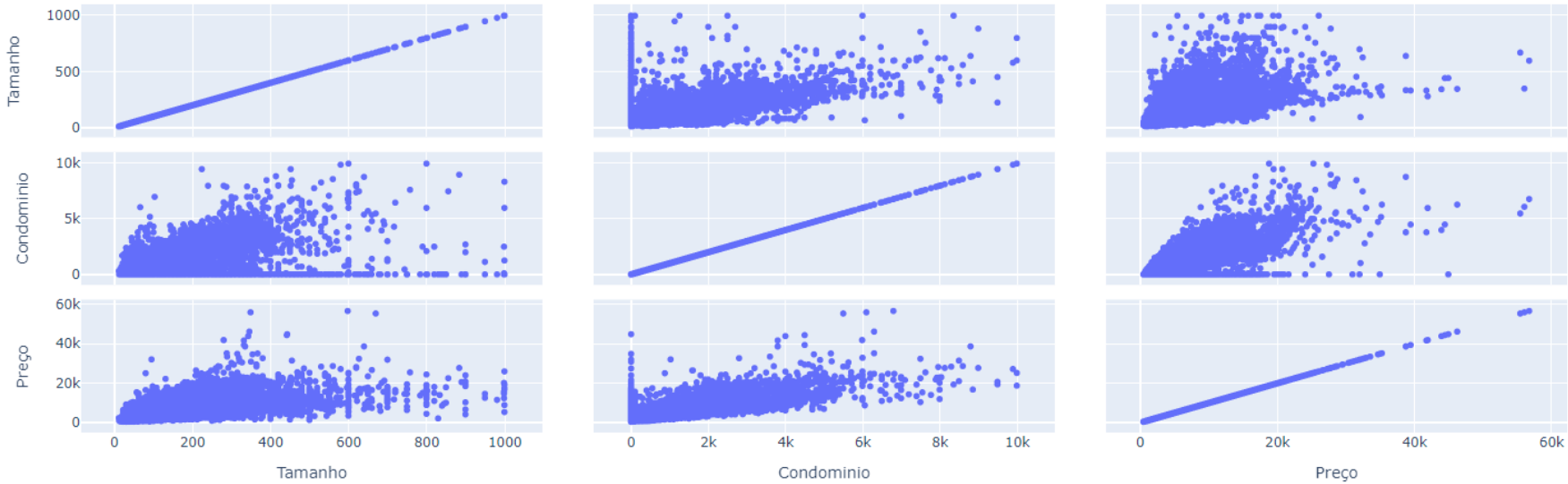
# Análise e Exploração dos Dados

```
#Analise estatitica do df.
base_total.describe()
```

| | Tamanho | Quartos | Banheiros | Estacionamento | Mobiliado | Condominio | Preço |
|---|---|---|---|---|---|---|---|
| **count** | 17884.000000 | 17884.000000 | 17884.000000 | 17884.000000 | 17884.000000 | 17884.000000 | 17884.000000 |
| **mean** | 119.510904 | 2.419425 | 2.177868 | 1.539588 | 0.215164 | 875.872456 | 4658.366249 |
| **std** | 109.926064 | 1.037972 | 1.249217 | 1.339713 | 0.410948 | 1002.256489 | 4360.982563 |
| **min** | 11.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 499.000000 |
| **25%** | 54.000000 | 2.000000 | 1.000000 | 1.000000 | 0.000000 | 295.000000 | 1911.000000 |
| **50%** | 76.000000 | 2.000000 | 2.000000 | 1.000000 | 0.000000 | 580.000000 | 3078.000000 |
| **75%** | 145.000000 | 3.000000 | 3.000000 | 2.000000 | 0.000000 | 1100.000000 | 5650.000000 |
| **max** | 1000.000000 | 9.000000 | 9.000000 | 8.000000 | 1.000000 | 10000.000000 | 56800.000000 |

# Análise e Exploração dos Dados

```
#Visualização dos dados
grafico = px.scatter_matrix(base_total, dimensions=['Tamanho', 'Condominio', 'Preço'])
grafico.show()
```



# Divisão entre Previsores e Classe

```
[ ]  #Previsores
     X_total = base_total.iloc[:, 0:6].values
```

```
[ ]  X_total
```

```
array([[4.700e+01, 2.000e+00, 2.000e+00, 1.000e+00, 0.000e+00, 2.200e+02],
       [4.500e+01, 2.000e+00, 2.000e+00, 1.000e+00, 0.000e+00, 1.480e+02],
       [4.800e+01, 2.000e+00, 2.000e+00, 1.000e+00, 0.000e+00, 1.000e+02],
       ...,
       [7.000e+01, 3.000e+00, 3.000e+00, 0.000e+00, 1.000e+00, 9.800e+02],
       [1.200e+02, 2.000e+00, 2.000e+00, 2.000e+00, 1.000e+00, 1.585e+03],
       [8.000e+01, 2.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00]])
```

```
[ ]  type(X_total)
```

```
numpy.ndarray
```

```
[ ]  #Classe
     y_total = base_total.iloc[:,6].values
```

```
[ ]  y_total
```

```
array([ 1150.,  1148.,  1100., ...,  7390., 14020.,  1587.])
```

```
[ ]  type(y_total)
```

```
numpy.ndarray
```

# Divisão de bases de Treinamento e Teste

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] X_total_treinamento, X_total_teste, y_total_treinamento, y_total_teste = train_test_split(X_total, y_total, test_size = 0.25, random_state = 42)
```

```
[ ] #Tamanho de cada vetor
    X_total_treinamento.shape, y_total_treinamento.shape, X_total_teste.shape, y_total_teste.shape
```

```
((13413, 6), (13413,), (4471, 6), (4471,))
```

# Aplicação de Algoritmos de Machine Learning

- Regressão Linear Múltipla;
- SVM (Máquina de vetores de suporte);
- Redes Neurais Artificiais;
- Gradient Boosting

# Aplicação de Algoritmos de Machine Learning

O processo para todos foi o mesmo, começando pela importação do respectivo algoritmo, realizando o treinamento por meio da função "fit" nas bases de treinamento e registrando sua performance por meio da função score, que calcula o coeficiente de determinação, $R^2$, tanto da base de treinamento como de teste.
Também utilizamos a função mean-absolute-error para calcular o erro médio absoluto de previsões no valor do aluguel.

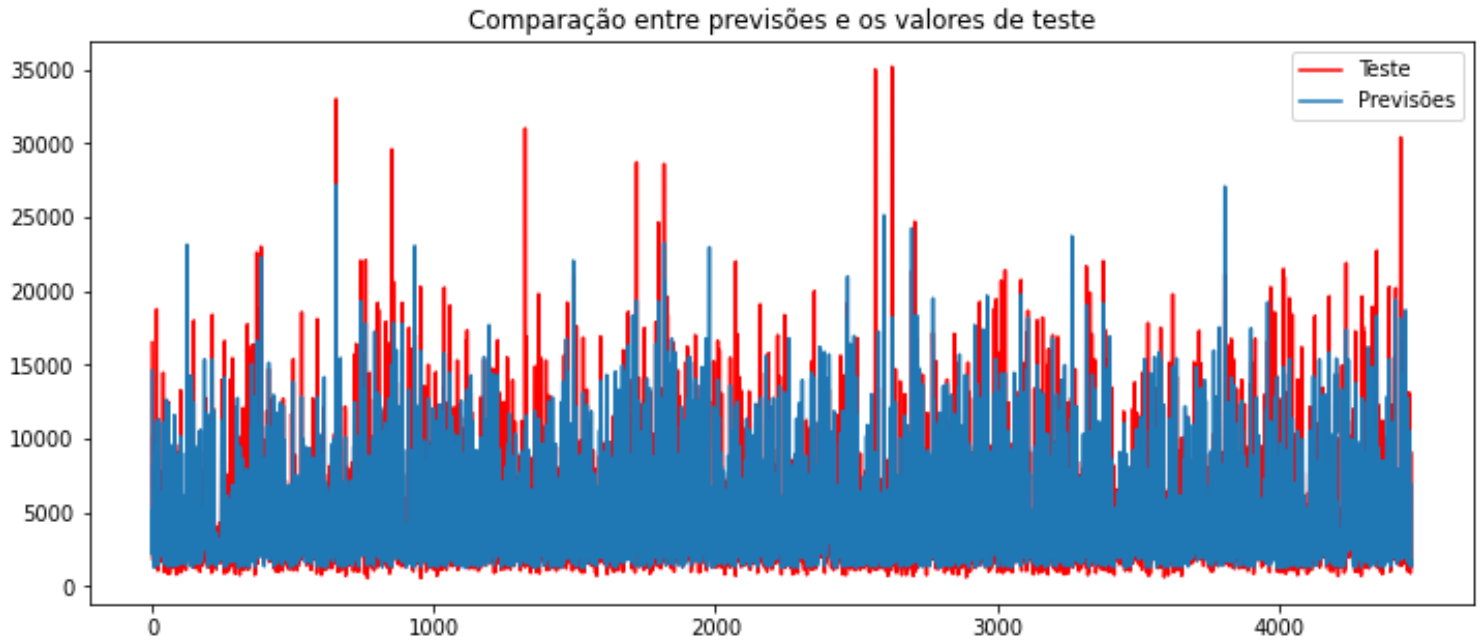
# Interpretação/Apresentação dos Resultados

| Algoritmo | Score base de treinamento | Score base de teste | Erro médio absoluto |
|---|---|---|---|
| Regressão linear múltipla | 0.71 | 0.73 | 1312 |
| SVM | 0.75 | 0.74 | 1217 |
| Redes Neurais | 0.73 | 0.74 | 1263 |
| RGB | 0.77 | 0.74 | 1256 |

# Interpretação/Apresentação dos Resultados

```
plt.figure(figsize=(12,5))
plt.plot(count, y_total_teste, color ="red")
plt.plot(count, previsoes)
plt.title("Comparação entre previsões e os valores de teste")
plt.gca().legend(('Teste','Previsões'))

plt.show()
```



# Cross-Validation

```
ApplyesKFold(X_total, y_total)
```

```
Linear Regression Média (R^2): 0.7163081629599922
svr Média (R^2): 0.7214209915149702
rna Média (R^2): 0.7298406252503852
rgb Média (R^2): 0.7330188157418 84
O melhor modelo é: rgb com valor: 0.7330188157418 84
```