**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**

**Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data**

**Alencar Lucas Pletsch**

**Previsão de Preços de Apartamentos em Curitiba-PR**

*(Bairros Selecionados)*

Belo Horizonte

2021

**Alencar Lucas Pletsch**

**Previsão de Preços de Apartamentos em Curitiba-PR**

*(Bairros Selecionados)*

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte

2021 **SUMÁRIO**

[1. Introdução 4](#_Toc70666760)

[1.1. Contextualização 4](#_Toc70666761)

[1.2. O problema proposto 4](#_Toc70666762)

[2. Coleta de Dados 8](#_Toc70666763)

[3. Processamento/Tratamento de Dados 14](#_Toc70666764)

[3.1. *Join* dos *dataframes* 14](#_Toc70666765)

[3.2. *Tratamento de Missing Values* 16](#_Toc70666766)

[3.3. *Dados Duplicados* 18](#_Toc70666767)

[3.4. *Tratamento de outliers* 18](#_Toc70666768)

[4. Análise e Exploração dos Dados 34](#_Toc70666769)

[4.1 Análise da correlação entre atributos 43](#_Toc70666770)

[4.2 Problema da dupla contagem de quartos e suítes 45](#_Toc70666771)

[5. Criação de Modelos de Machine Learning 49](#_Toc70666772)

[5.1 Ordinary Least Squares (Statsmodel) 50](#_Toc70666773)

[5.2 Ordinary Least Squares (Scikit-learn) 52](#_Toc70666774)

[5.3 Regularized Regression Methods - Ridge Regression (Scikit-learn) 54](#_Toc70666775)

[5.4 Least Absolute Shrinkage And Selection Operator regularization - LASSO (Scikit-learn) 56](#_Toc70666776)

[5.5 Elastic Net (Scikit-learn) 58](#_Toc70666777)

[5.6 Linear Support Vector Regression - SVM-LinearSVR (Scikit-learn) 60](#_Toc70666778)

[5.7 Stochastic Gradient Descent: Regressor - SGDRegressor (Scikit-learn) 62](#_Toc70666779)

[6. Apresentação dos Resultados 65](#_Toc70666780)

[6.1 Workflow da análise de dados 65](#_Toc70666781)

[6.2 Comparativo dos modelos 66](#_Toc70666782)

[6.3 *Dashboard* 68](#_Toc70666783)

[7. Links 80](#_Toc70666784)

[APÊNDICE 81](#_Toc70666785)

# 1. Introdução

## Contextualização

É de conhecimento geral que o mercado imobiliário possui grande importância na economia brasileira como um todo e, em especial, na vida íntima das pessoas, afinal a aquisição da “casa própria” é o sonho de muitas delas.

Por isso, foi escolhido um tema diretamente relacionado a esse assunto, qual seja precificação de ativos imobiliários (apartamentos) por meio de análise de regressão linear. Embora seja uma técnica simples, a análise de regressão linear tem grande aplicabilidade prática para a previsão de preços com base em dados de mercado.

No trabalho em questão foram utilizadas duas bases de dados, uma com os dados de apartamentos[[1]](#footnote-1) de alguns bairros da cidade de Curitiba-PR, e, outra com dados econômicos dos bairros, da mesma cidade, extraída do estudo intitulado “*Perfil demográfico e socioeconômico dos bairros agregados de Curitiba*”[[2]](#footnote-2), elaborado por meio de uma parceria entre o Departamento Intersindical de Estatística e Estudos Socioeconômicos (DIEESE) e a Prefeitura de Curitiba.

Espera-se que, após o tratamento e análise adequada, tal conjunto de dados possa servir para prever os preços de apartamentos em Curitiba-PR com certo grau de acerto, de modo a evitar transações por preços que fogem ao padrão normal, reduzindo assim a possibilidade de prejuízos financeiros às partes envolvidas.

## 1.2. O problema proposto

Nesse trabalho realizaremos uma análise dos dados referentes a preços de imóveis (apartamentos) na cidade de Curitiba-PR para fins de precificação. Na sequência, por meio da técnica dos 5-Ws, detalharemos o problema para melhor compreensão e delimitação do tema.

1. **Por que este problema é importante (why)?**

A importância desse problema repousa no impacto em termos de segurança e satisfação que a aquisição de um imóvel próprio proporciona as pessoas em geral. Os imóveis sempre estiveram no imaginário das pessoas como “um sonho a ser conquistado”, muito disso em função do valor dispendido para a sua aquisição. Como a renda média das pessoas é baixa, muitas delas levam a vida inteira para adquirir um imóvel, caso obtenham êxito nessa empreitada.

Enquanto buscam esse objetivo, convivem com riscos relacionados ao não pagamento de aluguel, como despejo e inscrição em cadastros de inadimplentes, riscos relacionados à redução da renda em função da necessidade de pagamento de aluguel, riscos relacionados à perda de liberdade em sentido amplo nos casos de moradia conjunta com familiares etc. Esses riscos podem afetar a autoestima das pessoas causando-lhes danos à saúde como ansiedade e depressão, e, também, desestabilizar a harmonia familiar, motivando em muitos casos agressões e divórcios com danos emocionais para filhos.

Na ânsia de se livrar desses riscos muitas pessoas entram no mercado imobiliário demandando imóveis sem ter, entretanto, a expertise necessária para fazer a correta avaliação do preço ofertado. Nesses casos, em função do desconhecimento natural e da urgência na aquisição do imóvel, são facilmente influenciáveis com os discursos já conhecidos de que “*nunca se perde dinheiro com imóveis”* e de que *“adquirir imóveis é um investimento”*, acarretando compras desvantajosas e prejuízos financeiros que podem expor as pessoas aos mesmos riscos que pretendiam evitar.

Do lado dos ofertantes de imóveis, precificar adequadamente evitaria influências de corretores e imobiliárias tendentes a fechar a negociação no menor prazo possível a fim de receber o quanto antes a comissão que lhe é devida. Também evitaria o risco de incorrer no conceito de ilusão monetária e achar que está ganhando dinheiro quando, realmente não está, pois muitas pessoas não dispõem do conhecimento necessário para efetuar cálculos de atualização do capital, considerando taxas de inflação e taxas de juros de referência.

1. **De quem são os dados analisados e como foram coletados (who)?**

Os dados foram obtidos de duas fontes distintas conforme abaixo:

***- Apartamentos à venda por bairro em Curitiba:*** Este conjunto de dados faz parte da disciplina “Manipulação e Visualização de Dados”, do Laboratório de Estatística e Geoinformação da Universidade Federal do Paraná (LEG/UFPR) e foi apresentado pelo Prof. Walmes M. Zeviani, em 10/06/2019, aos seus alunos como conjuntos de dados para praticar com a linguagem R[[3]](#footnote-3). Segundo consta do enunciado da questão “*São dados de apartamentos à venda em Curitiba anunciados em site de imóveis. As informações são de imóveis em 7 bairros de Curitiba e descrevem cada imóvel em termos de preço de venda, metragem e número de cômodos*.”

***- Área em Km2, população e habitantes por Km2, segundo bairro agregado:*** Este conjunto de dados consta do Anexo I do estudo intitulado “*Perfil demográfico e socioeconômico dos bairros agregados de Curitiba*”[[4]](#footnote-4), elaborado por meio de uma parceria entre o Departamento Intersindical de Estatística e Estudos Socioeconômicos (DIEESE) e a Prefeitura de Curitiba.

1. **Qual é o objeto desta análise (what)?**

Por meio desse trabalho analisaremos os dados de preços de imóveis (apartamentos) de sete bairros da cidade de Curitiba-PR, bem como dados referentes ao perfil demográfico (densidade populacional) para fins de precificação.

Para tanto, utilizaremos os algoritmos de regressão linear simples disponíveis nas bibliotecas Scikit-learn[[5]](#footnote-5) e Statsmodel[[6]](#footnote-6).

1. **Em que ambiente geográfico esta análise está inserida (where)?**

O escopo dessa análise está restrito aos conjuntos de dados que se referem à parte dos bairros de Curitiba-PR, ou seja, não estão disponíveis os dados de todos os bairros da cidade.

1. **Qual o período compreendido nesta análise (when)?**

Com relação ao período analisado, verificou-se que os dados obtidos no estudo do DIEESE em parceria com a Prefeitura de Curitiba, embora realizado em fevereiro de 2016, referem-se ao Censo/IBGE de 2010.

Já os dados de preço de apartamento obtido do LEG/UFPR, embora a atividade possua como data o ano de 2019, não há informação sobre o período da coleta dos dados. Como o arquivo de texto com os dados possui o nome *“ap\_venda7bairros\_cwb\_210314.txt”*, deduz-se que seja de março de 2014.

Apesar de haver um hiato temporal entre os dados do censo de 2010 e a coleta de preços de 2014, acreditamos que tenha se mantido a proporção da densidade populacional entre os bairros distintos, não afetando o resultado da análise.

# 2. Coleta de Dados

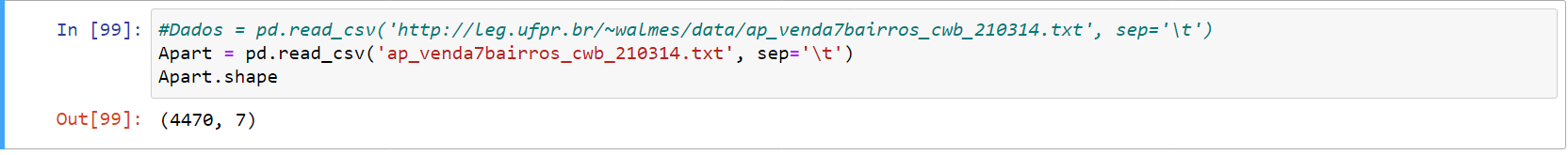
Os dados utilizados na presente análise são oriundos de duas bases diferentes, uma com os preços de imóveis (apartamentos) de sete bairros de Curitiba-PR e outra com dados demográficos da mesma região. A seguir apresentaremos maiores detalhes de cada base de dados.

***2.1. Apartamentos à venda por bairro em Curitiba (Apartamentos):***

Como já foi mencionado anteriormente, este conjunto de dados faz parte da disciplina “Manipulação e Visualização de Dados”, do Laboratório de Estatística e Geoinformação da Universidade Federal do Paraná (LEG/UFPR) e foi apresentado pelo Prof. Walmes M. Zeviani, em 10/06/2019, aos seus alunos como conjuntos de dados para praticar com a linguagem R[[7]](#footnote-7). Segundo consta do enunciado da questão “*São dados de apartamentos à venda em Curitiba anunciados em site de imóveis. As informações são de imóveis em 7 bairros de Curitiba e descrevem cada imóvel em termos de preço de venda, metragem e número de cômodos*.”

Estes dados foram obtidos no site da disciplina supracitada no seguinte endereço eletrônico: [*leg.ufpr.br/~walmes/data/ap\_venda7bairros\_cwb\_210314.txt*](http://leg.ufpr.br/~walmes/data/ap_venda7bairros_cwb_210314.txt), em 05/01/2021. Como pode ser percebido, trata-se de um arquivo de texto (.txt), sendo importado para o Jupyter Notebook por meio da biblioteca Pandas[[8]](#footnote-8) usando os seguintes comandos:

Figura Código para leitura do dataset



No caso foi optado pelo download do arquivo e posterior leitura pelo Pandas, pois havia o receio de que a página não estivesse disponível em algum momento futuro. Porém, poderia ter sido feita a leitura do dataset diretamente na página em que está hospedado por meio do comando que está comentado (#).

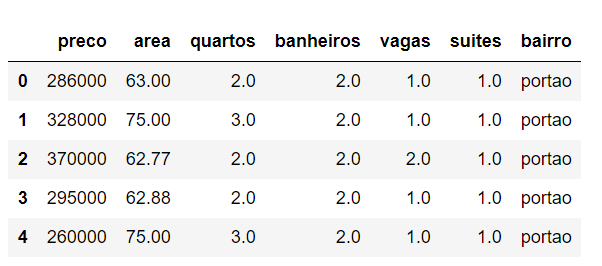
O dataset possui o formato de 4.470 linhas por 7 colunas e está estruturado da seguinte forma:

Tabela Estrutura do dataset Apartamentos

| **Nome da coluna/campo** | **Descrição** | **Tipo** |
| --- | --- | --- |
| preco | Valor anunciado para venda do apartamento | int64 |
| area | Área do apartamento em metros quadrados | float64 |
| quartos | Quantidade de quartos incluindo suítes | float64 |
| banheiros | Quantidade de banheiros incluindo banheiros das suítes | float64 |
| vagas | Quantidade de vagas para estacionamento de veículos | float64 |
| suites | Quantidade de suítes (quartos com banheiros) | float64 |
| bairro | Nome do bairro em que está localizado o apartamento | object |

Abaixo segue o *head* do *dataframe* do Pandas:

Figura - Head do dataframe Apartamentos



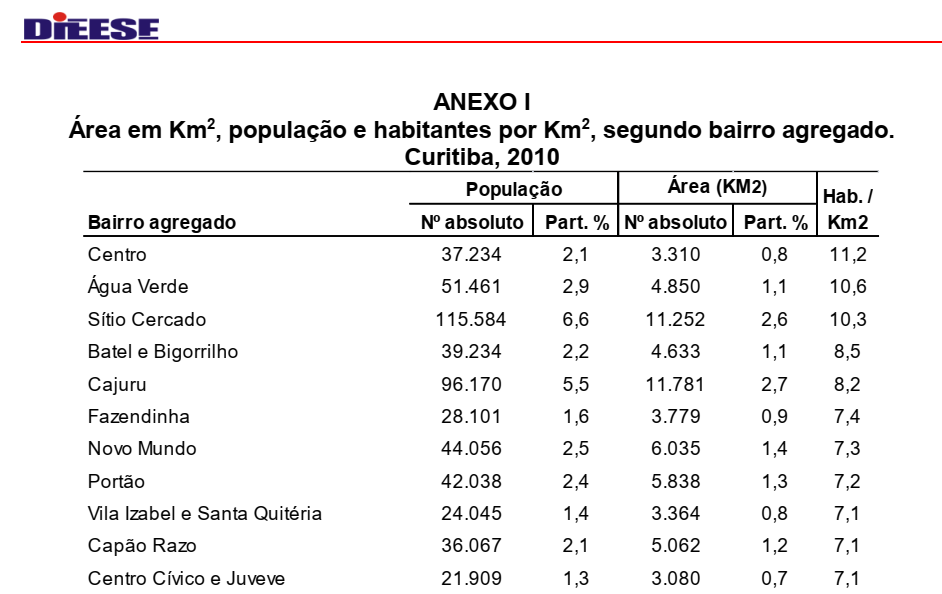
O campo que será utilizado para o *join* dos *dataframes* é o denominado “bairro”. Assim, a relação entre os *datasets* se dá pelo atributo bairro.

***2.2. Área em Km², população e habitantes por Km², segundo bairro agregado (Bairros)***

Retomando o já mencionado anteriormente, este conjunto de dados consta do Anexo I do estudo intitulado “*Perfil demográfico e socioeconômico dos bairros agregados de Curitiba*”[[9]](#footnote-9), elaborado por meio de uma parceria entre o Departamento Intersindical de Estatística e Estudos Socioeconômicos (DIEESE) e a Prefeitura de Curitiba. Embora o estudo seja de fevereiro de 2016, referem-se ao Censo/IBGE de 2010.

Tal estudo foi obtido na internet em 07/01/2021, no endereço eletrônico [*bairros.pdf (coreconpr.gov.br)*](http://www.coreconpr.gov.br/wp-content/uploads/2016/07/bairros.pdf), página do Conselho Regional de Economia do Paraná. Diferentemente do *dataset* anterior, este é uma tabela intitulada Anexo I dentro de um arquivo PDF (vide figura abaixo).

Figura - Anexo I do estudo do DIEESE



Para a leitura dos dados foi utilizada a biblioteca Tabula[[10]](#footnote-10), a qual permite ler os dados em arquivos PDF e convertê-los em *dataframes* do Pandas. Abaixo segue fragmento do código utilizado para leitura e manipulação do *dataframe*, manipulação esta necessária pois na leitura inicial o dataframe ficou desfigurado.

Figura - Leitura e manipulação do dataset Bairros



Após o tratamento acima demonstrado o *dataframe* ficou com o formato de 40 linhas e 6 colunas, sendo todos as colunas tipo *object.* Desse modo, foram necessários mais alguns ajustes no dataframe como correção de nomes[[11]](#footnote-11) e transformações dos tipos de atributos conforme código abaixo:

Figura - Transformação do dataframe Bairros



Na figura acima podemos observar o *head* do *dataframe* Bairros. Na tabela abaixo vemos a estrutura dos campos do *dataframe* Bairros:

Tabela Estrutura do dataframe Bairros

| **Nome da coluna/campo** | **Descrição** | **Tipo** |
| --- | --- | --- |
| Bairro | Nome do bairro agregado em que está localizado o apartamento | object |
| População | População do bairro agregado | int64 |
| População(%) | População percentual do bairro agregado | float64 |
| Área(km2) | Área do bairro agregado | int64 |
| Área(%) | Área percentual do bairro agregado | float64 |
| Habitantes/km2 | Densidade populacional do bairro agregado | float64 |

Como pode ser percebido acima, o *dataset* Bairros utiliza o conceito de “Bairro Agregado” e refere-se à junção de alguns bairros circunvizinhos de Curitiba-PR para a realização do estudo.

Entendemos que essa variação do conceito não prejudicará o resultado da análise, sendo, então, considerado como se fosse o mesmo bairro do *dataset* Apartamento.

O relacionamento dos dois *datasets* será por meio dos atributos ‘bairro’ (dataframe = Apart) e ‘Bairro’ (dataframe = Bairros), os quais basicamente tem o nome dos bairros de Curitiba-PR. No próximo tópico discorreremos sobre os comandos utilizados para a realização da junção dos *dataframes.*

# 3. Processamento/Tratamento de Dados

Embora já tenhamos no tópico anterior trabalhado parcialmente os conjuntos de dados para fins de leitura e estruturação dos *dataframes*, agora vamos tratar da parte de processamento e tratamento dos dados a partir da junção dos *dataframes*.

## 3.1. *Join* dos *dataframes*

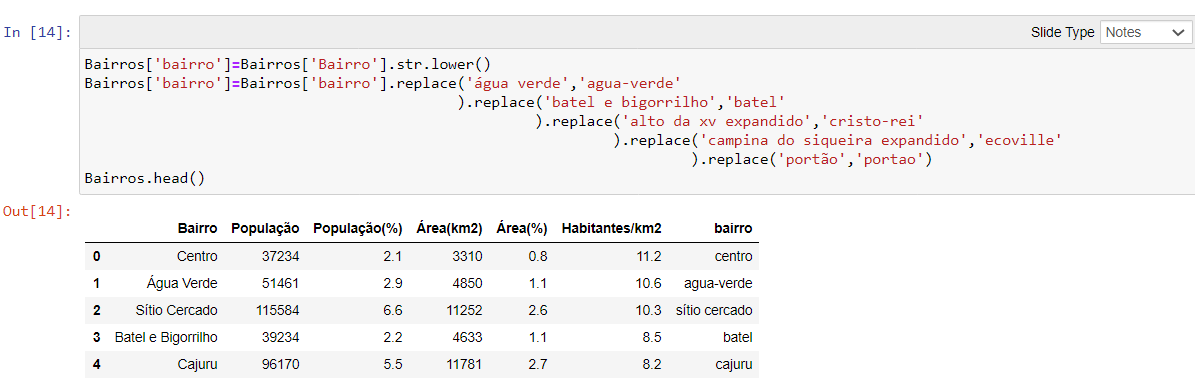
Primeiro, relembramos que faremos o *join* pelas colunas que contém o nome dos bairros em ambos os *datasets*. Verificamos que há diferença na descrição dos bairros em cada *dataset*, o que inviabilizaria a correspondência adequada dos registros.

Figura - Descrição dos nomes dos bairros em cada dataset



Dessa forma, foi necessário criar a coluna “bairro” no dataset “Bairro” e ajustar a descrição dos bairros para haver a correspondência exata com a coluna “bairro” do dataset “Apart”, o que foi realizado conforme o código a seguir:

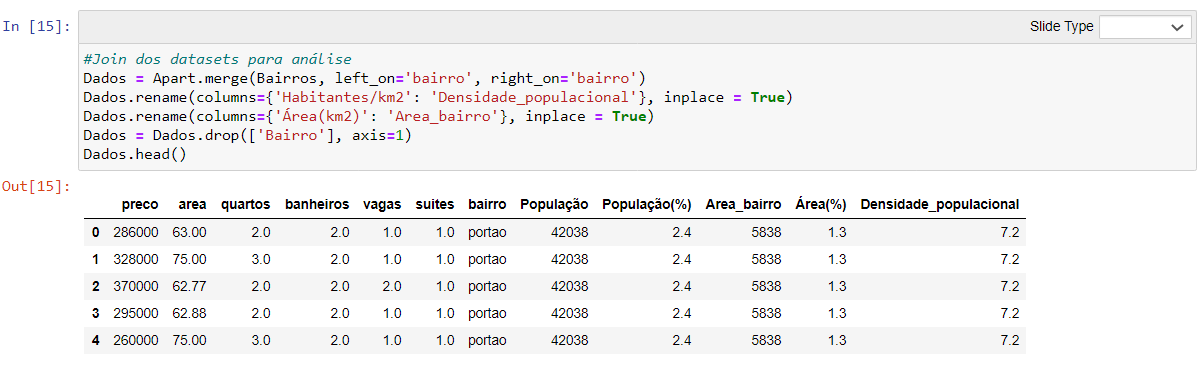
Figura - Criação da coluna "bairro" e ajustes dos registros com o nome dos bairros



Na sequência foi realizado o join dos dataframes “Apart” e “Bairros” em um novo dataframe denominado “Dados”. Sendo, posteriormente, alterado o nome das colunas “Habitantes/km2” e “Área(km2)” para “Densidade\_populacional” e “Area\_bairro”, respectivamente. Tal alteração dos nomes foi necessária para evitar erros nos algoritmos de regressão executados posteriormente.

Também, foi eliminada a coluna “Bairro” pois esta não seria mais de utilidade para a análise. Seguem os códigos:

Figura - Join dos Dataframes



Depois foi acrescentada a coluna “preco\_metro” com o valor do cálculo do preço/m² e convertido o tipo de dados da coluna “bairro” para “category” em função de erros na execução do algoritmo de regressão linear.

Figura - Inclusão da coluna "preco\_metro" e mudança do tipo de dados da coluna "bairro"



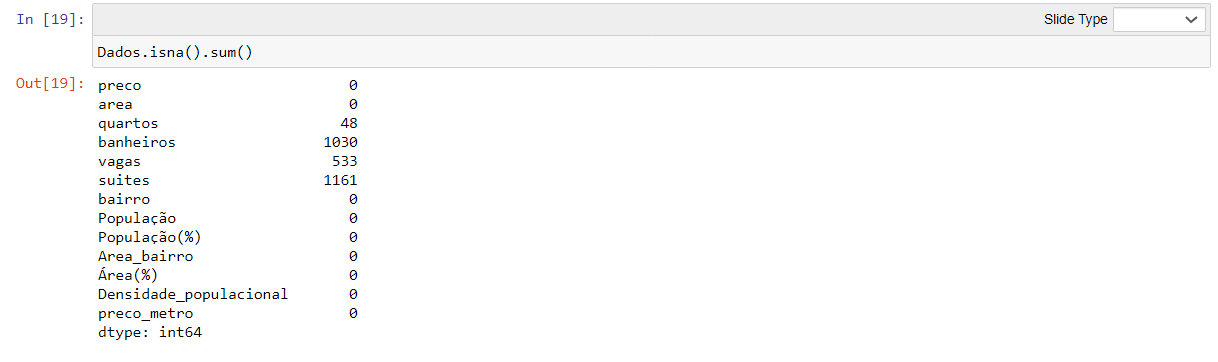
## 3.2. *Tratamento de Missing Values*

Com os datasets consolidados em um só vamos analisar os valores faltantes (*missing values*) e fazer o tratamento adequado.

Verificamos no dataset “Dados” uma grande quantidade de missing values, os quais são oriundos do dataset “Apart” que continha dados de anúncios. Provavelmente, a ausência desses valores seja decorrente de anúncios preenchidos de forma equivocada e que prejudica em demasia a análise dos dados.

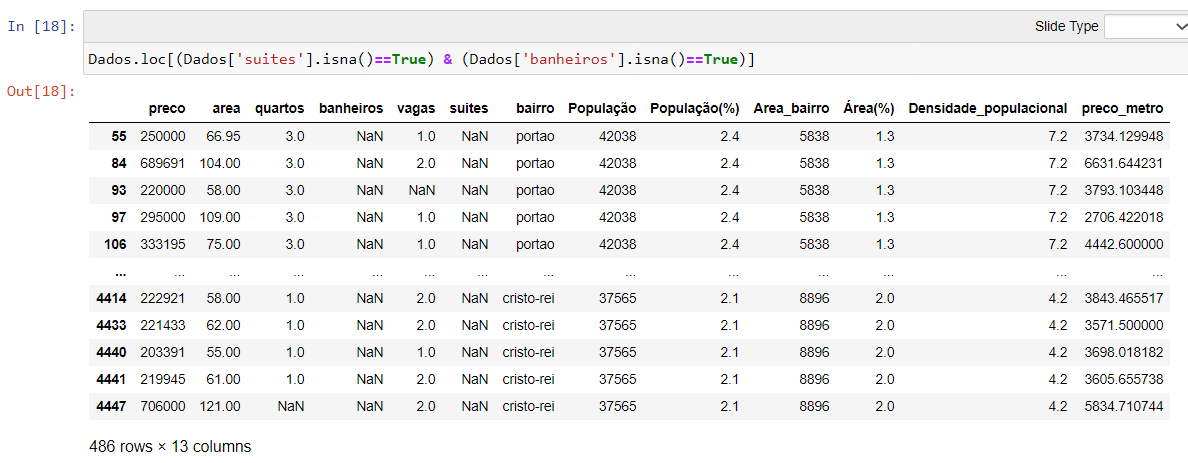
Abaixo podemos verificar a quantidade de missing values por campos dos registros:

Figura - Missing values



Analisando os registros com missing values tanto na coluna “suites” quanto na coluna “banheiros”, as quais são as colunas com maior quantidade de dados ausentes, contabilizamos 486 registros.

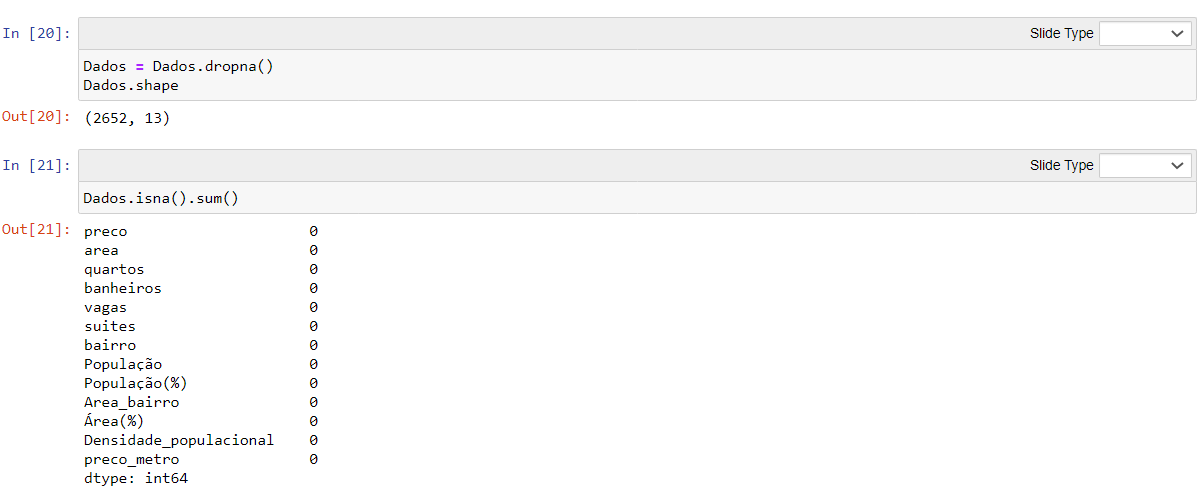
Figura - Dados missing nas colunas "banheiros" e "suites", simultaneamente



Apesar de haver técnicas que sugerem a substituição dos missing values por valores como média, mediana etc., entendemos que, ao fazer isso, estaríamos distorcendo de tal maneira os registros que poderiam ocorrer aberrações como apartamentos com 1 quarto e três banheiros.

Assim, dado que, mesmo excluindo os registros com missing values, ainda teríamos uma quantidade relevante de registros para análise (2.652 registros) preferimos manter a integridade destes e evitar distorções que possam prejudicar o resultado da análise. Abaixo segue o código utilizado para exclusão dos missing values.

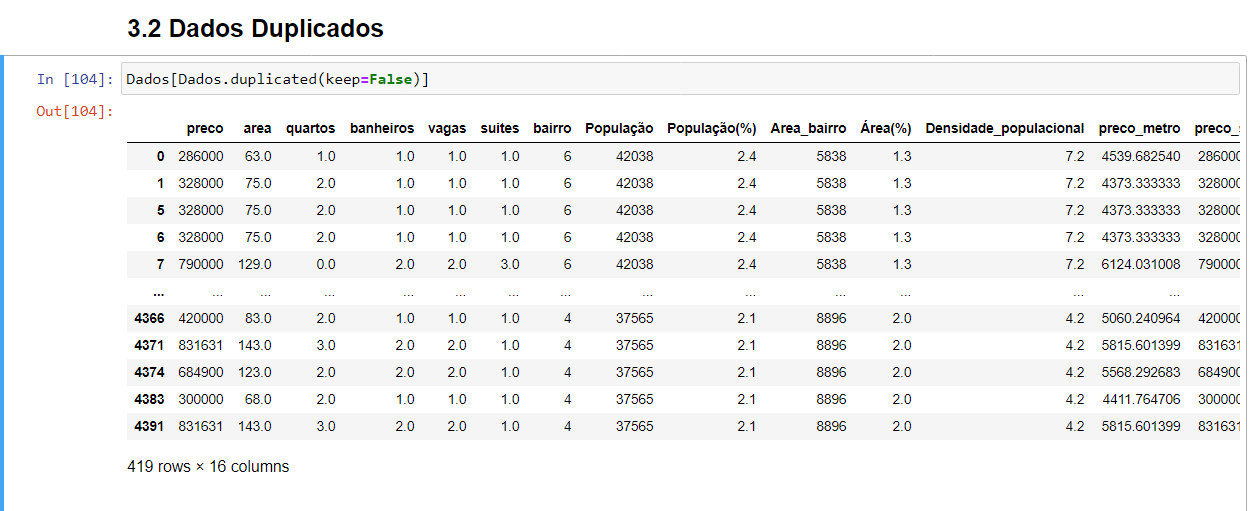
Figura - Exclusão de missing values



## 3.3. *Dados Duplicados*

Após análise dos dados duplicados, verificamos a existência de 419 registros nessa situação.

Figura - Dados duplicados



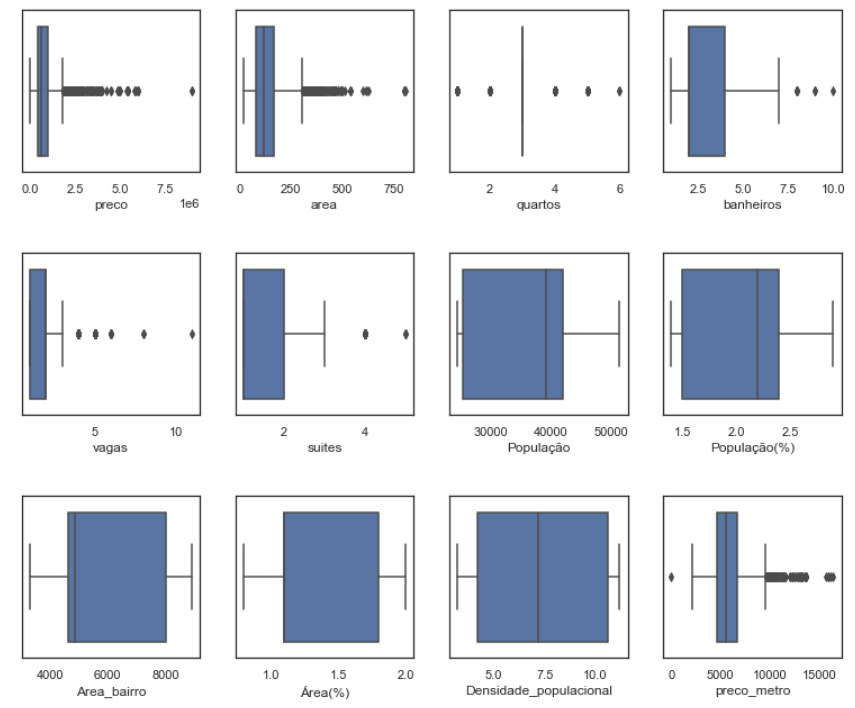
Embora em alguns casos seja possível verificar que o registro duplicado seja referente ao mesmo imóvel, essa não é a regra presente no dataset. Como são anúncios é comum que o vendedor renove a publicação com certa frequência, mas, também, pode ocorrer que alguns apartamentos diferentes tenham as mesmas características, inclusive o preço. Isso ocorre principalmente nos anúncios de menor valor, os quais se comportam como se fossem “comodities” em função da ausência de diferenciação.

Assim, em função da dificuldade de distinção de qual registro seria decorrente de duplicatas ou decorrente de anúncios idênticos, e, para não perder mais registros, optou-se por manter as duplicatas no dataset para fins de análise.

## 3.4. *Tratamento de outliers*

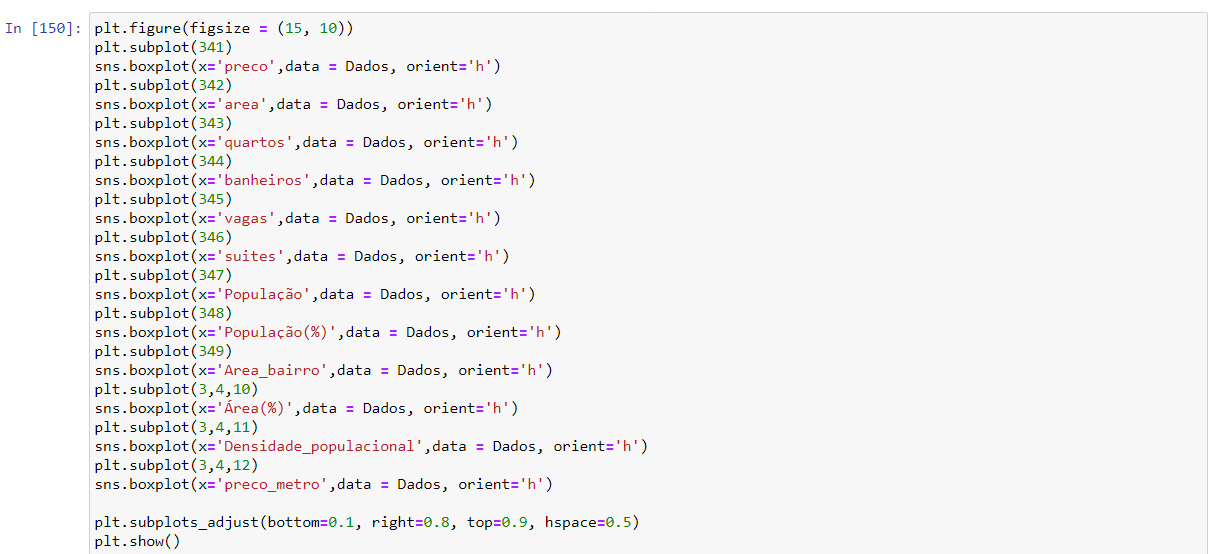
Agora vamos partir para a análise de outliers. Nos boxplots abaixo, gerados por meio das bibliotecas matplotlib.pyplot[[12]](#footnote-12) e seaborn[[13]](#footnote-13), podemos verificar a existência de outliers nas variáveis “preco”, “area”, “quartos”, “banheiros”, “vagas”, “suítes” e “preco\_metro”.

Figura - Boxplot atributos do dataframe Dados



O código para gerar os boxplot pode ser visualizado abaixo.

Figura - Código para gerar boxplot

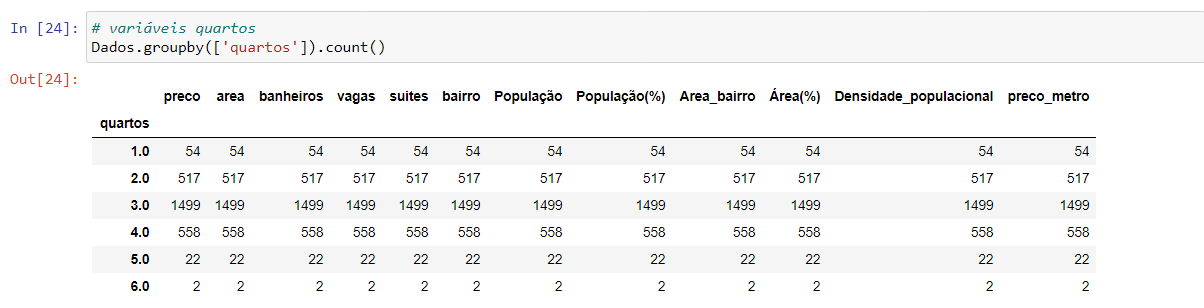


Como já exposto anteriormente, optou-se por modificar o mínimo possível o conjunto de dados para evitar distorções. Assim, vamos verificar inicialmente as variáveis quartos, banheiros, vagas e suítes.

1. **Quartos:**

Por meio do Pandas podemos agrupar os dados pela variável “quartos” e obter a contagem dos valores:

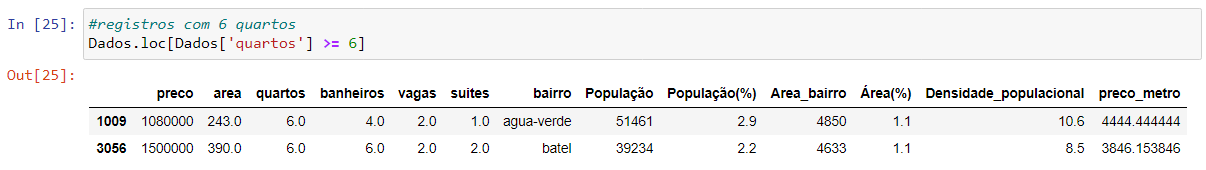
Figura - Contagem de apartamentos pelo número de quartos



Pelo boxplot, apenas os apartamentos com três quartos não são classificados como outliers. Isso se deve à predominância de apartamentos com essa quantidade de quartos. Entretanto, como a base já está reduzida, não podemos desprezar os apartamentos com menos de seis quartos.

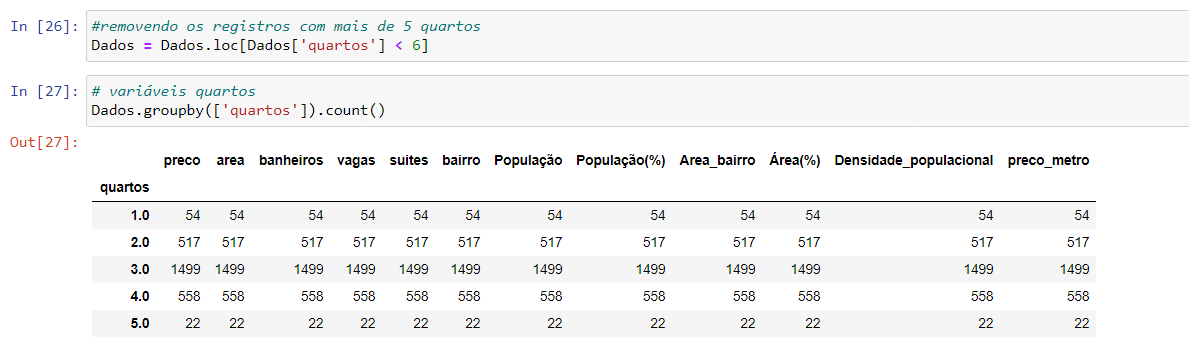
Já com 6 quartos temos dois registros, vamos analisá-los:

Figura - registros com 6 quartos



Como são apenas dois registros vamos removê-los da base para reduzir as distorções.

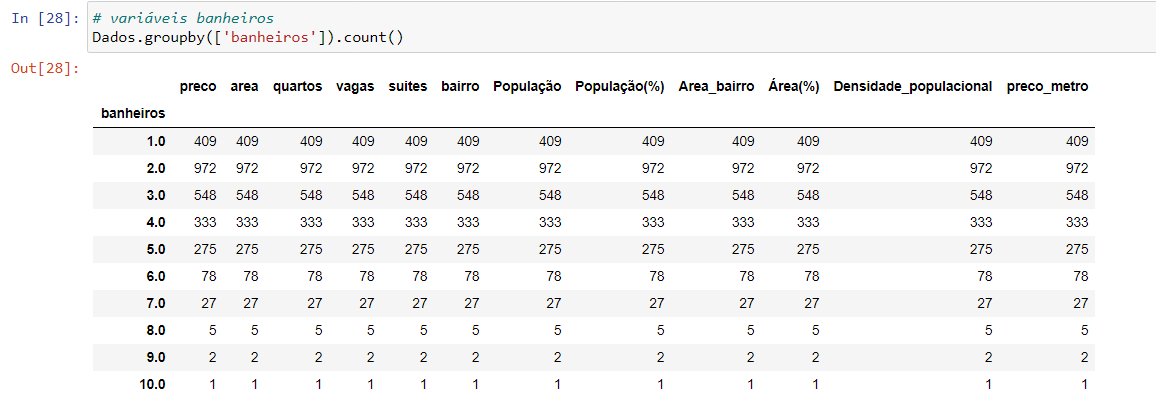
Figura - Remoção dos registros com mais de 6 quartos



1. **Banheiros:**

Novamente, por meio do Pandas vamos agrupar a variável “banheiros” e obter a contagem dos valores:

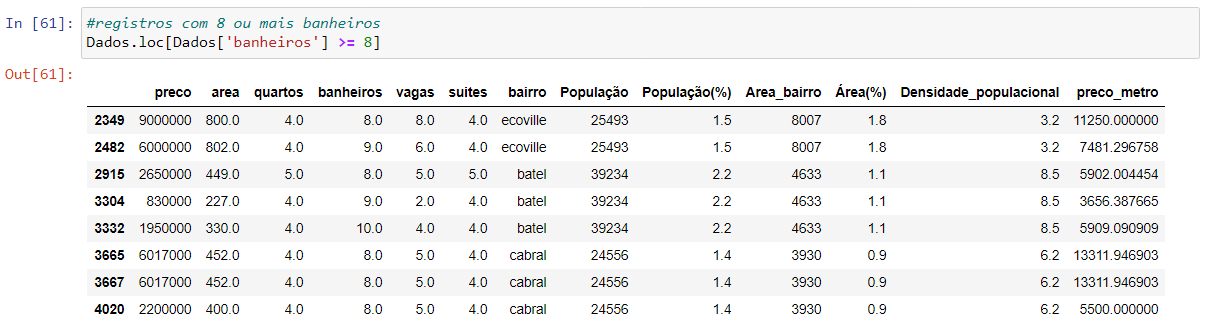
Figura - Contagem de apartamentos pelo número de banheiros



Pelo boxplot, apenas os apartamentos com menos de sete banheiros não são classificados como outliers. Mais uma vez, como a base já está reduzida, não podemos desprezar os apartamentos com menos de oito banheiros.

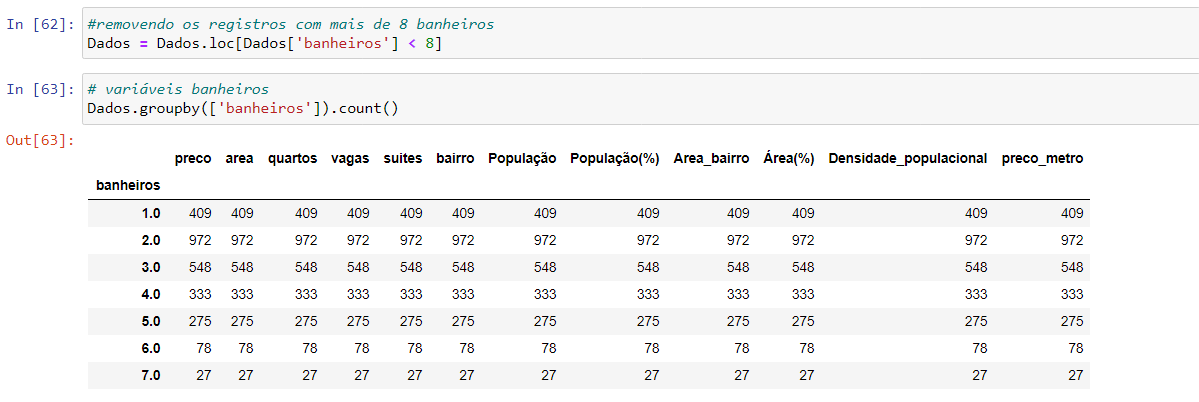
Já com oito ou mais banheiros temos oito registros, vamos analisá-los:

Figura - registros com 8 ou mais banheiros



Como são apenas oito registros vamos removê-los da base para reduzir as distorções.

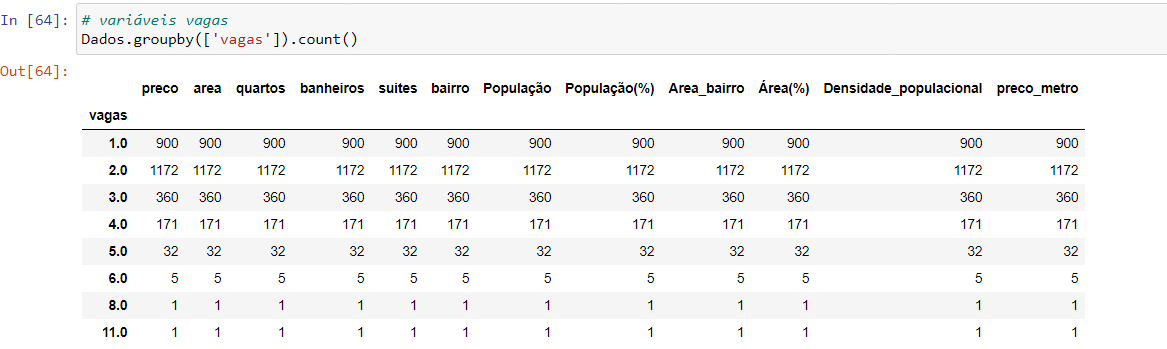
Figura - Remoção dos registros com 8 ou mais banheiros



1. **Vagas:**

Agora vamos agrupar a variável “vagas” e obter a contagem dos valores:

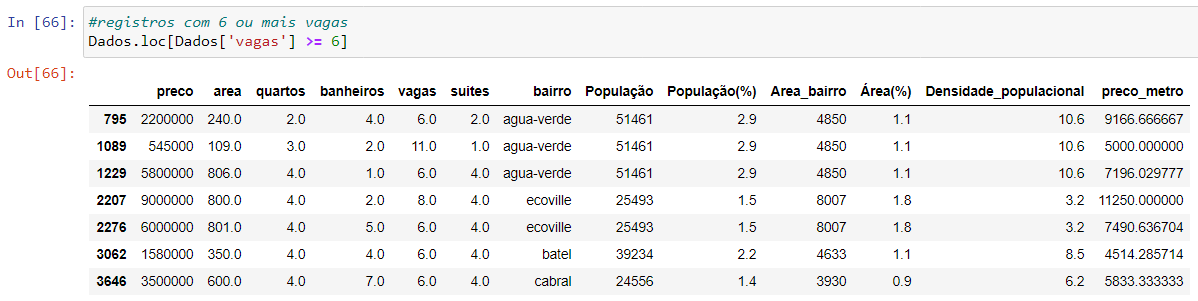
Figura - Contagem de apartamentos pelo número de vagas



Pelo boxplot, apenas os apartamentos com menos de quatro vagas não são classificados como outliers. Como há muitos registros com quatro e cinco vagas, não podemos desprezar os apartamentos com menos de 6 vagas.

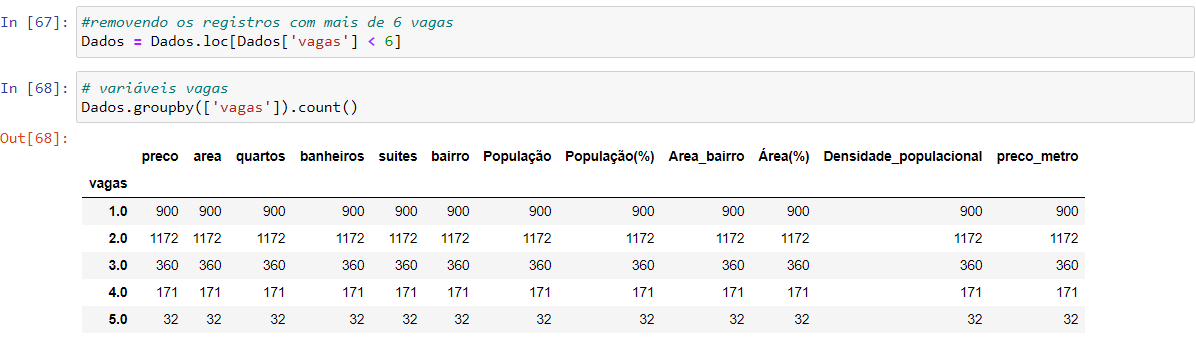
Já com 6 ou mais vagas temos sete registros, vamos analisá-los:

Figura - registros com 6 ou mais vagas



Como são apenas sete registros vamos removê-los da base para reduzir as distorções.

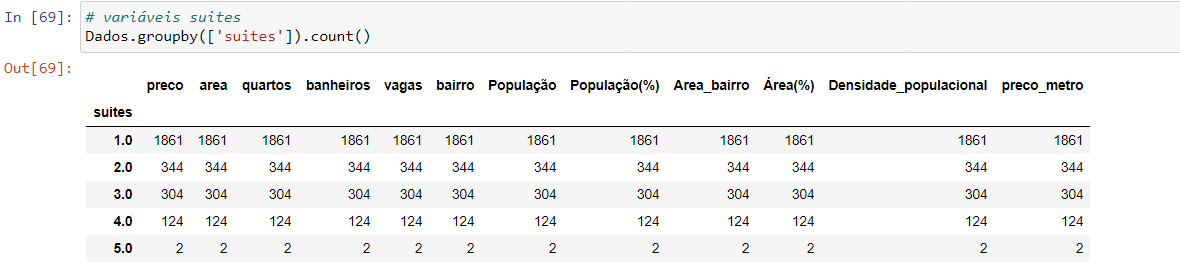
Figura - Remoção dos registros com 6 ou mais vagas



1. **Suítes:**

Agora vamos agrupar a variável “suites” e obter a contagem dos valores:

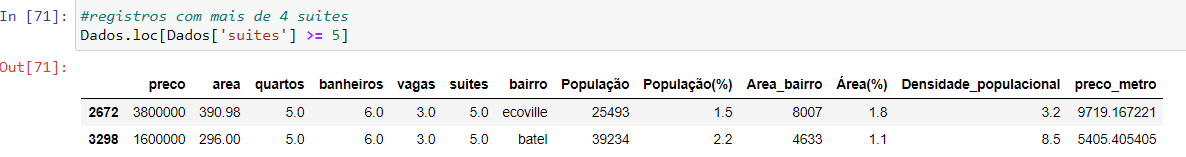
Figura - Contagem de apartamentos pelo número de suítes



Pelo boxplot, apenas os apartamentos com menos de quatro suítes não são classificados como outliers. Como há muitos registros com quatro suítes, não podemos desprezar os apartamentos com menos de 5 suítes.

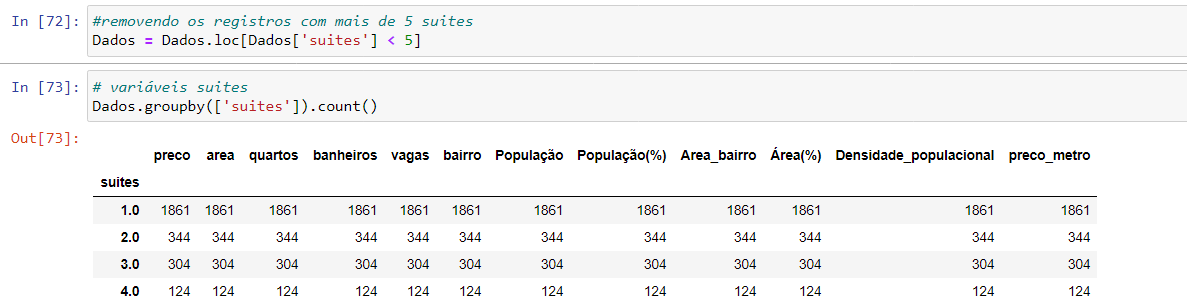
Já com 5 ou mais suítes temos dois registros, vamos analisá-los:

Figura - registros com 5 ou mais suites



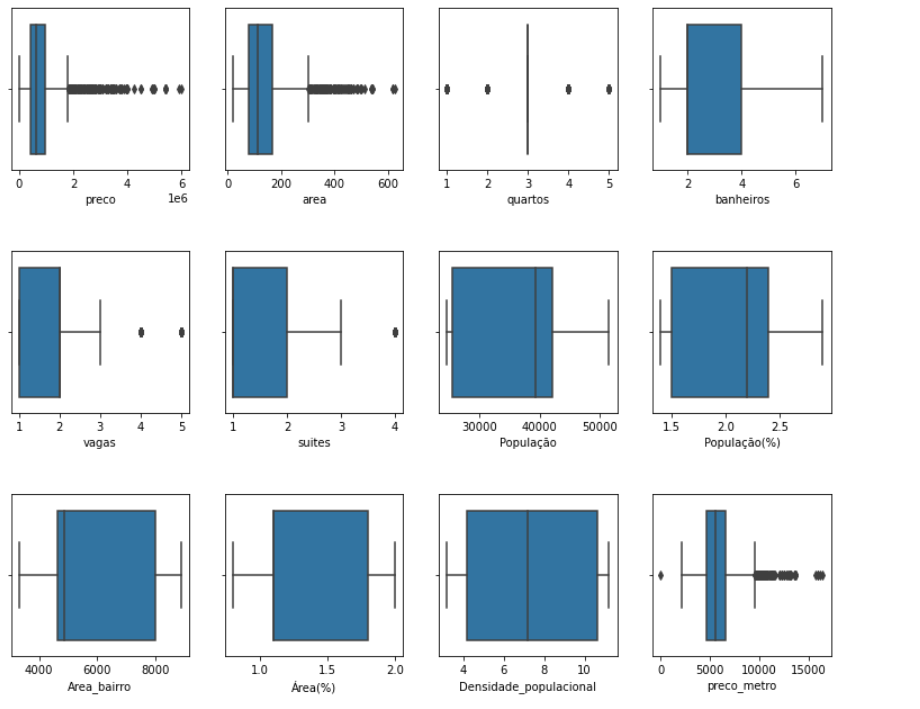
Como são apenas dois registros vamos removê-los da base para reduzir as distorções.

Figura - Remoção dos registros com 5 ou mais vagas



Após a remoção de 19 registros poderemos visualizar novamente os boxplots e analisar os outliers.

Figura - Boxplot atributos do dataframe Dados após remoção de outliers



Verifica-se acima que houve uma redução substancial dos outliers das variáveis tratadas, sendo que os outliers que ainda permanecem no dataset não podem ser removidos sob pena de empobrecer demasiadamente a base de dados.

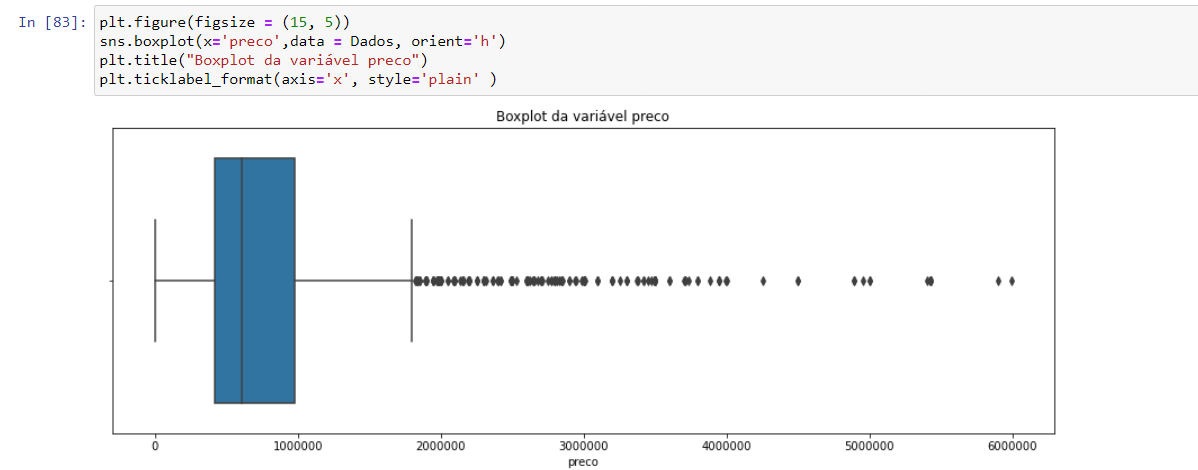
Com relação à variável “área” ainda há muitos outliers mas não será feito nenhum tratamento para não distorcer a base de dados. A variável “preco\_metro” é consequência das variáveis “preco” e “area” e não será tratado diretamente.

Já a variável “preco” ainda apresenta grande quantidade de outlier, impactando a variável “preco\_metro”. Para a variável “preco” apresentaremos uma forma alternativa para tratamento de outliers.

1. **Preço:**

Vamos na sequência visualizar de forma ampliada o boxplot da variável “preco”:

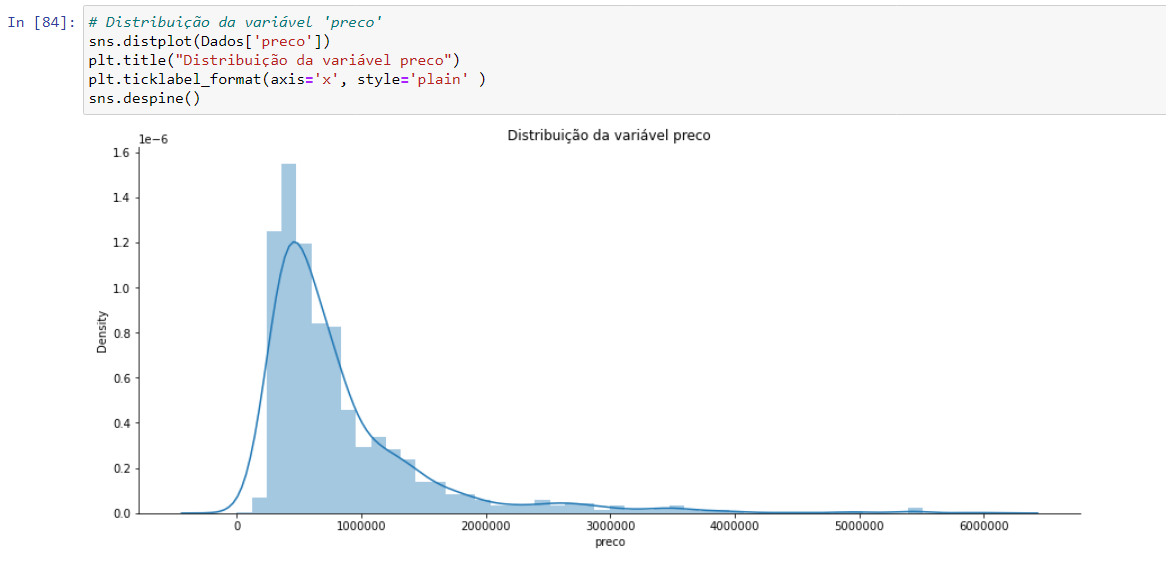
Figura - Boxplot da variável "preco"



Percebe-se a concentração de registros até próximo de R$ 200 mil por unidade. A partir desse valor temos muitos outliers que precisam ser tratados.

Abaixo temos a distribuição da variável visto em um gráfico de densidade no qual podemos perceber a cauda longa à direita onde estão situados os outliers.

Figura - Gráfico de densidade da variável preço



Para tratar esse outliers vamos utilizar uma abordagem diferenciada. Não vamos substituí-los com valores de mediana, média etc., vamos substituí-los com os valores preditos obtidos por meio de um modelo preliminar.

Basicamente, vamos seguir os seguintes passos:

- cálculo dos limites inferiores e superiores dos outliers por meio do IQR (distância interquartílica);

- criação de um modelo preliminar com a biblioteca Statsmodel;

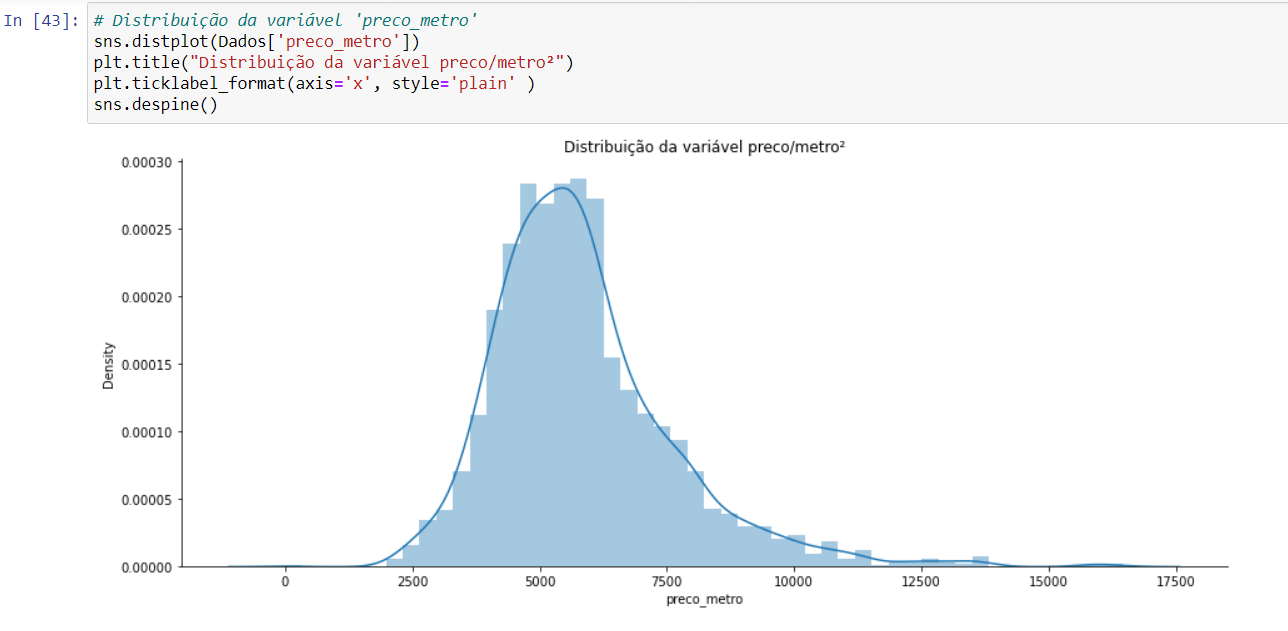
- criação de uma coluna com os preços preditos pelo modelo preliminar e outra para receber os valores originais, no caso de não outlier, e valores preditos, no caso de que o original seja outlier;

- criação de uma coluna com um marcador de outlier (sim ou não); e

- criação de uma coluna com preços sem outliers (substituição dos preços dos registros marcados com outlier “sim” pelos preços preditos).

Mas antes de seguir estes passos vamos verificar a variável “preco\_metro”. Observando o gráfico abaixo vemos que os valores de preços/m² estão concentrados acima de R$ 2.000,00/m².

Figura - Distribuição da variável "preco\_metro"



Filtrando os valores abaixo de R$ 2.000,00/m² temos apenas um registro.

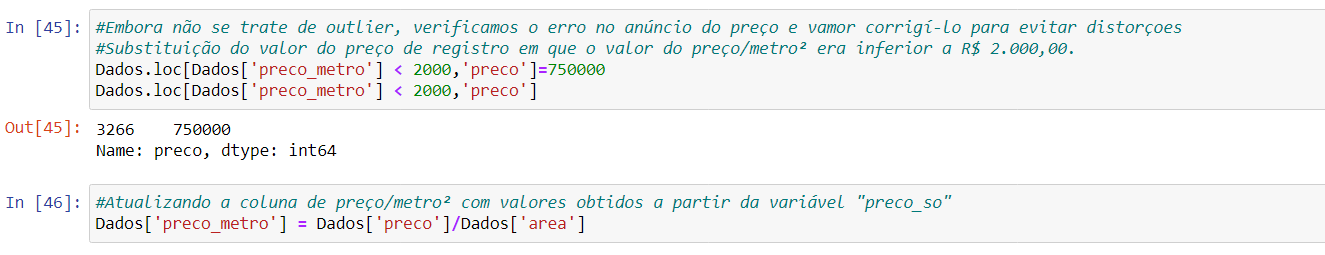
Figura - Valor abaixo de R$ 2.000,00/m²



Percebe-se que houve um equívoco no preenchimento do anúncio, sendo o valor do preço do apartamento registrado por R$ 750,00. Estimamos que, dado os demais atributos, o valor correto seria R$ 750 mil e aí procedemos à substituição desse valor antes de continuar o tratamento dos outliers.

Também realizamos a atualização da coluna “preco\_metro”. Segue o código para conferência.

Figura - Substituição do valor anunciado incorretamente e atualização da variável "preco\_metro"



Agora sim, seguindo os passos informados mais acima, iniciaremos pelo cálculo da distância interquartílica (IQR)[[14]](#footnote-14). Por meio da biblioteca Numpy[[15]](#footnote-15) vamos calcular os percentis 25 e 75, os quais são equivalentes ao 1º (Q1) e 3º quartil (Q3), respectivamente. Em seguida vamos subtrair o 1º quartil do 3º obtendo a distância interquartílica (IQR).

Após isso, vamos calcular os limites superiores e inferiores para a detecção de outliers. O limite inferior é obtido subtraindo do Q1 o valor de 1,5 vezes o valor de IQR. Já o limite superior é obtido adicionando ao Q3 o valor de 1,5 vezes o valor de IQR.

Os registros com preços de apartamentos inferiores ao limite inferior ou superior ao limite superior serão considerados outliers e receberão o adequado tratamento. Abaixo podemos observar o código para cálculo dos limites.

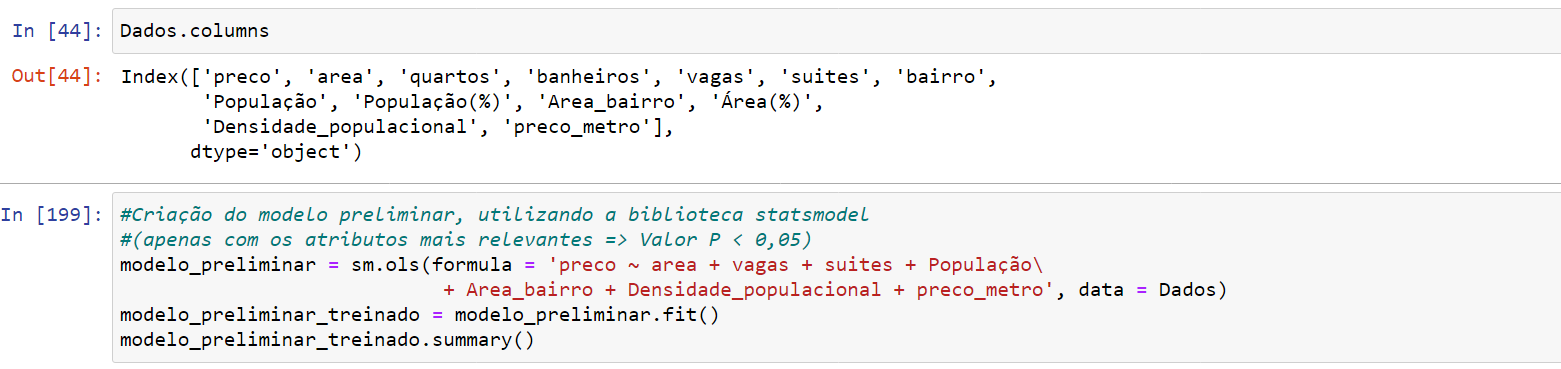
Figura - Cálculo dos limites para detecção de outliers



Na sequência faremos a criação do modelo preliminar, utilizando a biblioteca Statsmodel e o método de regressão linear de mínimos quadrados (OLS – Ordinary Least Squares)[[16]](#footnote-16). Para o modelo preliminar foram utilizadas apenas as variáveis mais relevantes, considerando para isso o Valor P (P value) abaixo de 0,05 e eliminando as variáveis menos relevante seguindo o processo conhecido como *Stepwise[[17]](#footnote-17).*

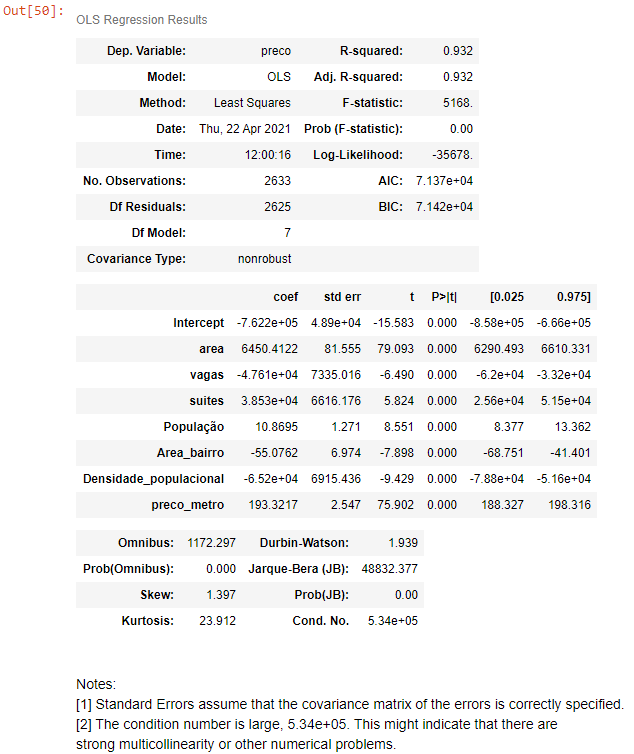
Desse processo, que teve como Coeficiente de Determinação (R²)[[18]](#footnote-18) o valor 0.932, resultaram como variáveis relevantes as seguintes: “area”, “vagas”, “suites”, “População”, “Area\_bairro”, “Densidade\_populacional”, “preco\_metro”. Abaixo temos o código desenvolvido.

Figura - Criação e ajuste do modelo preliminar



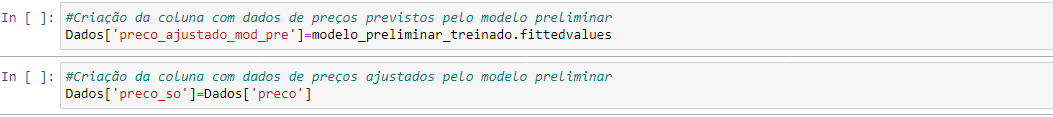
Abaixo podemos ver o sumário da modelo preliminar.

Figura - Resultado do modelo preliminar



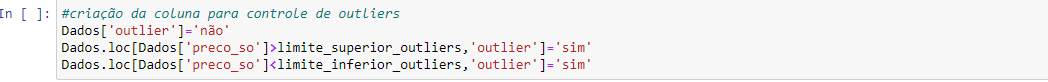
Continuando, temos a criação de uma coluna com os preços preditos pelo modelo preliminar e outra coluna que receberá parte dos valores originais (não outliers) e parte dos valores preditos (outliers). A primeira coluna será nomeada como “preco\_ajustado\_mod\_pre” e a segunda como “preco-so” (preço sem outlier). Segue o código.

Figura - Criação de colunas para receber preços preditos e preços sem outliers



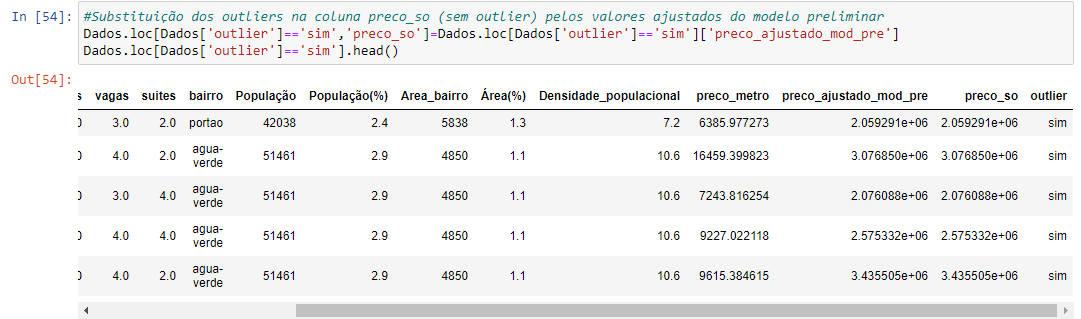
No próximo bloco de código vamos criar a coluna para controle de outliers, primeiramente preenchendo todos os registros com a *string* “não”, e, na sequência, substituindo os registros superiores ao limite superior ou inferiores ao limite inferior pela string “sim”. Dessa forma, todos os registros que contém *outliers* no atributo “preco” terão o valor “sim” na coluna “outlier”. Seguem os códigos:

Figura - Criação da coluna para controle de outliers



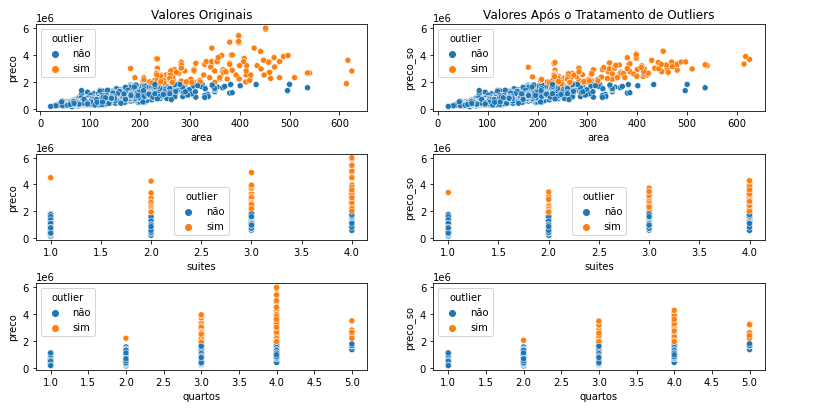
E por fim, faremos a substituição na coluna “preco\_so” (sem outlier) dos valores de preços com a correspondente marcação na coluna “outlier” com a *string* “sim” pelos valores constantes, dos mesmos registros, na coluna “preco\_ajustado\_mod\_pre”, conforme abaixo.

Figura - Substituição dos outliers na coluna "preco\_so"



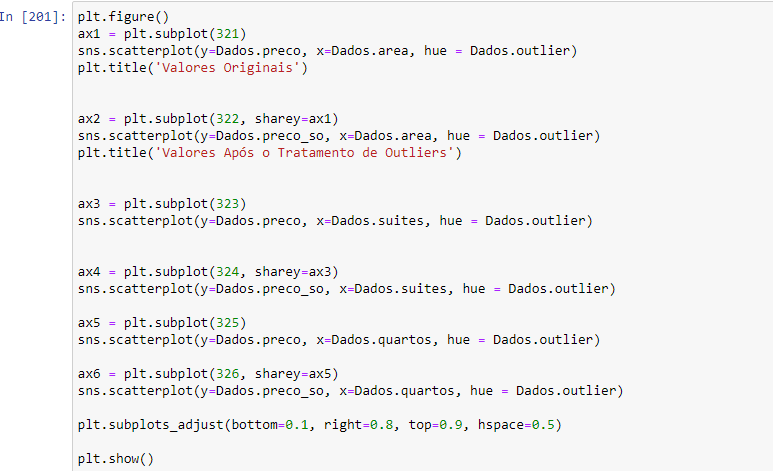
No gráfico abaixo podemos visualizar a alteração de preço antes e depois do tratamento de outliers. Ressalta-se que a coluna “preco” mantem os valores originais e a coluna “preco\_so” possui os dados tratados.

Figura - Valores de preço antes e depois do tratamento de outliers



Como se pode perceber acima, após o tratamento de outliers ocorreu uma menor dispersão dos valores, ficando mais concentrados próximo de uma linha de tendência. O código para construção do gráfico acima pode ser verificado abaixo:

Figura - Código para construção dos gráficos visualizados anteriormente



Agora vamos verificar na construção de um segundo modelo preliminar se houve uma melhora no R² após o tratamento de outliers na variável “preco” (no caso foi utilizado a variável “preco\_so”, a qual contém os preços sem outliers).

Figura - Criação do 2° modelo preliminar

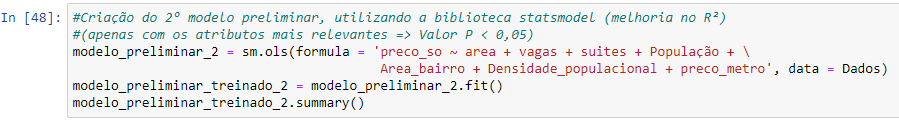
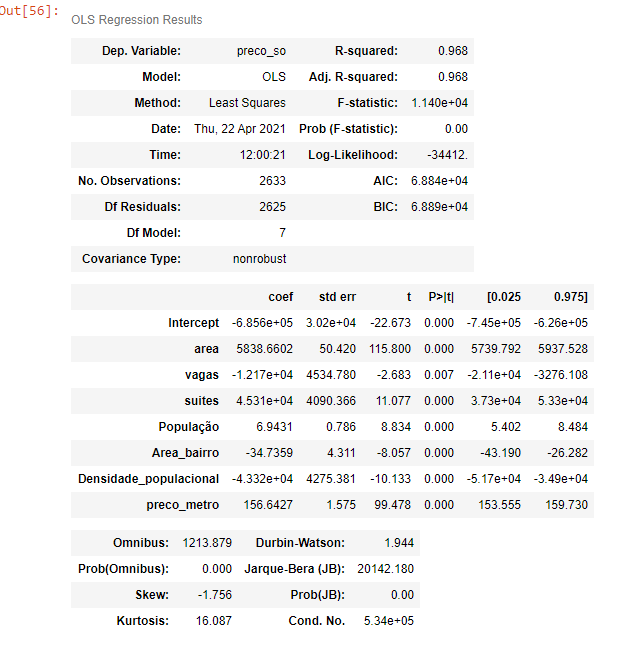


Figura - Resultado do 2° modelo preliminar



Vemos que houve uma melhora do Coeficiente de Determinação (R²) que antes era 0,932 e agora é 0,968, o que denota que estamos evoluindo na construção do modelo.

Embora não tenha sido realizado o tratamento de *outliers* com outras técnicas como média, mediana etc., entendemos que da forma como foi realizada há uma distorção menor dos dados, pois nesta técnica o peso de outros atributos como quartos, suítes etc., são considerados na hora de calcular os valores que serão utilizados para substituir os outliers.

# 4. Análise e Exploração dos Dados

Agora vamos analisar e explorar a base de dados em busca de padrões e/ou informações que possam ser úteis para melhor compreensão das relações existentes entre os diversos atributos dos apartamentos e bairros de Curitiba-PR constantes do dataset.

Primeiro, vamos observar as tabelas abaixo com as estatísticas do *dataset* “Dados” fornecidas pelo método *describe()* da biblioteca Pandas.

Tabela - Estatísticas do dataset Dados – parte 1

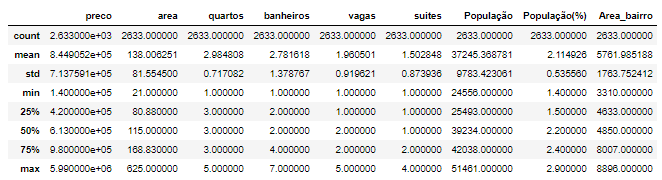


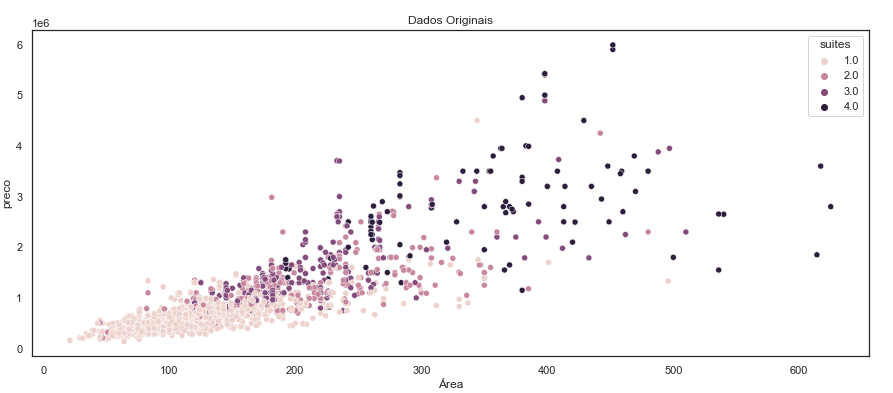
Tabela - Estatísticas do dataset Dados – parte 2



Vemos acima que os apartamentos possuem em média 138 m², cerca de 3 quartos, 2,78 banheiros, 1,96 vagas e 1,5 suítes.

A seguir, no gráfico abaixo temos a comparação do preço anunciado com a área do apartamento.

Figura - Preço x Área

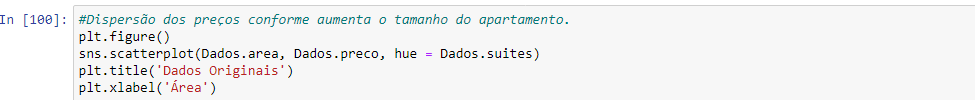


Nesse gráfico vemos nitidamente que ao passo que aumenta a área dos apartamentos os preços ficam mais dispersos. Isso pode denotar o incremento no grau de diferenciação dos imóveis em função de outros atributos que não são capturados no *dataset*, como qualidade dos materiais, idade do imóvel, necessidade de efetuar a venda em um curto prazo etc.

Nos apartamentos com área menor vemos uma maior concentração dos preços, significando o menor grau de diferenciação e a maior importância do atributo área para definição do preço. Pode-se inferir que, nessa região do gráfico, os imóveis se comportam como commodities, simplificando a comparação entre as unidades.

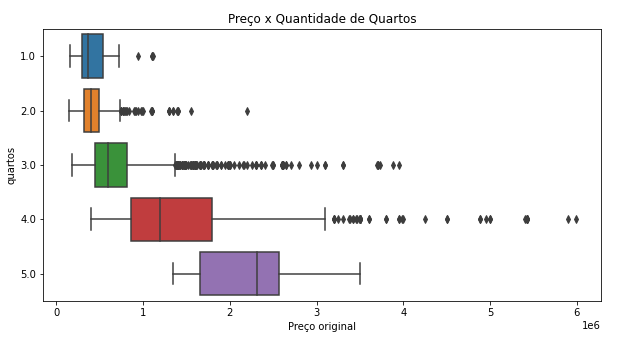
Segue o código para geração do gráfico:

Figura - código para geração do gráfico acima

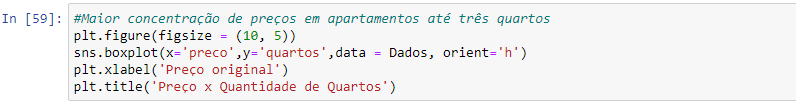


Comparando preços por quartos, podemos ver no gráfico abaixo que a concentração de valores ocorre em maior intensidade nos apartamentos de até três quartos.

Figura - Preços x Quartos



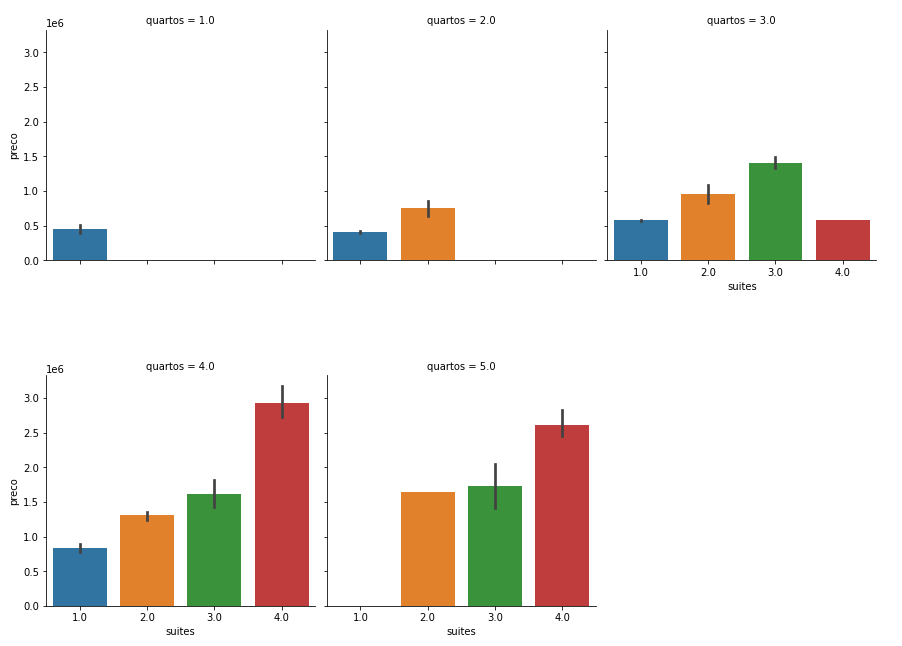
O código para gerar o gráfico acima pode ser visto a seguir.



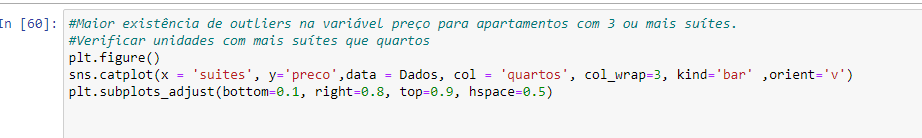
Quando comparamos quartos, suítes e preços podemos perceber que a dispersão maior nos valores ocorre nos apartamentos com mais de quatro quartos e três suítes.

Isso pode ser visto no comprimento da linha preta no topo de cada coluna. Quanto maior, mais dispersos são os dados.

Figura - Comparação quartos, suítes e preço



Segue o código para construção deste gráfico:



Percebemos no gráfico acima, mais um erro de anúncio. Na coluna mais acima e à direita temos uma unidade com três quartos e quatro suítes. Como isso pode distorcer os cálculos de regressão linear, vamos efetuar a correção, igualando a quantidade de suítes ao número de quartos. Abaixo o código para correção.

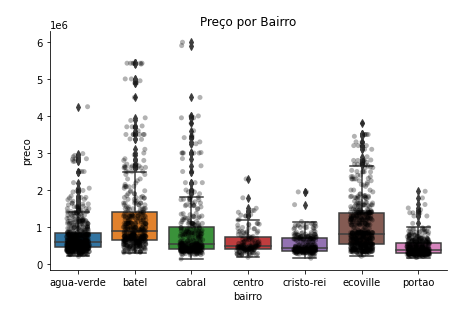


E conferindo novamente o gráfico vemos que o problema foi resolvido.

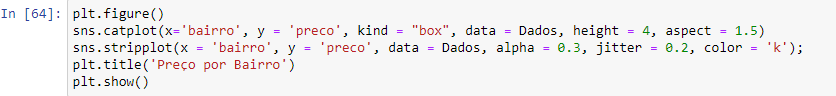
Figura - Comparativo de quartos, suítes e preço atualizado



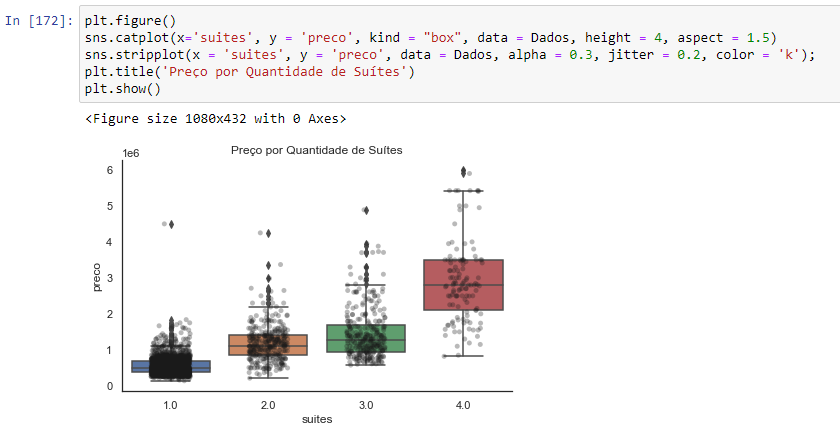
Agora, comparando preços por bairros, podemos ver no gráfico abaixo em quais bairros estão localizados aqueles apartamentos diferenciados, ou seja, maiores e com mais quantidades de quartos e suítes.

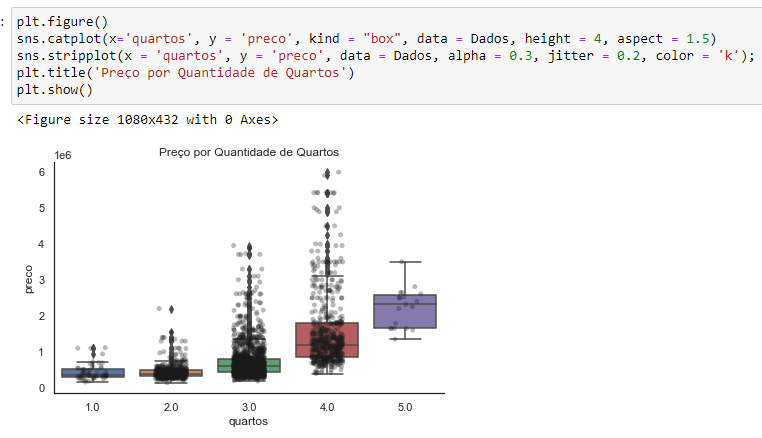


Na sequência, Cabral, Batel, Água-Verde e Ecoville são os bairros onde estão localizados os apartamentos com maior grau de diferenciação. Centro, Cristo-Rei e Portão são os bairros onde estão concentrados os apartamentos de menor valor. Segue o código para confecção do gráfico acima.



Agora com o mesmo gráfico acima (e respectivo código) vamos segregar por suítes, quartos, vagas e banheiros.



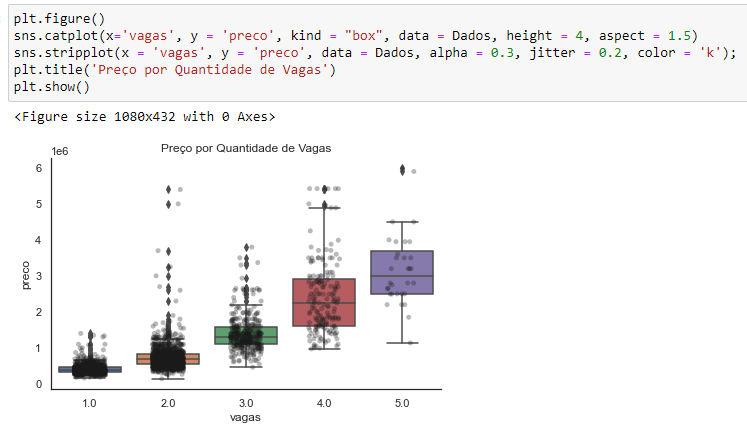


Comparando quartos com suítes vemos que, além da dispersão maior de valores para quantidades maiores de quartos ou suítes, com cinco quartos a regra é que praticamente todos sejam suítes.

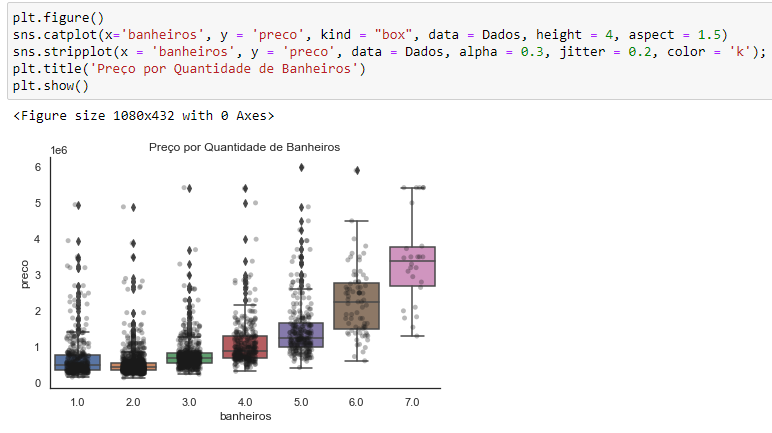
Tabela - Quantidade de quartos x suítes

| **quartos** | **1.0** | **2.0** | **3.0** | **4.0** | **5.0** |
| --- | --- | --- | --- | --- | --- |
| **suites** |  |  |  |  |  |
| **1.0** | 54.0 | 468.0 | 1193.0 | 146.0 | - |
| **2.0** | - | 48.0 | 52.0 | 240.0 | 4.0 |
| **3.0** | - | - | 253.0 | 48.0 | 4.0 |
| **4.0** | - | - | - | 112.0 | 11.0 |

Muita dispersão de valores também é percebido para apartamentos com mais de 3 vagas.



Com relação ao número de banheiros, a dispersão ocorre a partir de quatro banheiros.



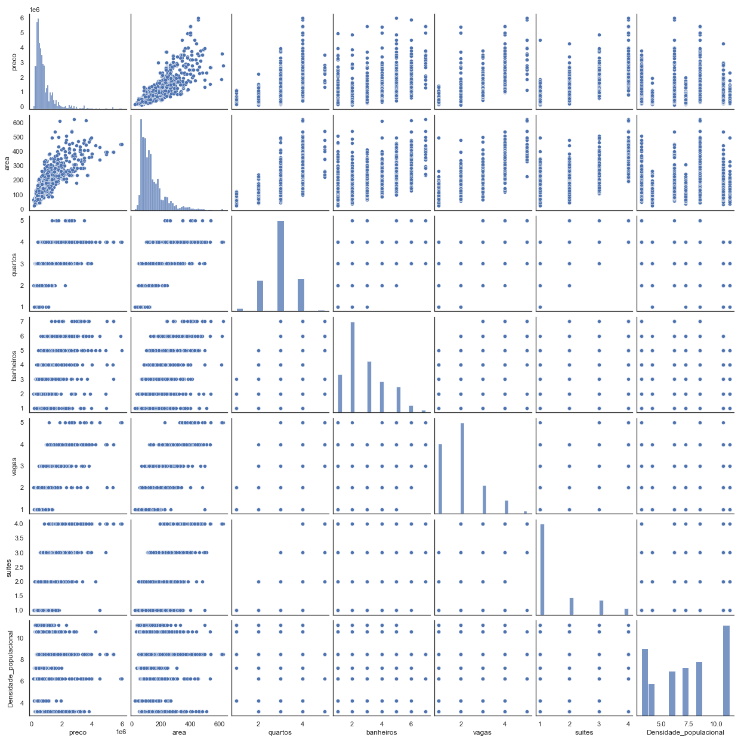
Na tabela abaixo verificamos a contagem de apartamentos por banheiros, quartos e suítes.

Tabela - Comparativo de banheiros, quartos e suítes

| **quartos** | **1.0** | **2.0** | | **3.0** | | | **4.0** | | | | **5.0** | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **suítes** | **1.0** | **1.0** | **2.0** | **1.0** | **2.0** | **3.0** | **1.0** | **2.0** | **3.0** | **4.0** | **2.0** | **3.0** | **4.0** |
| **banheiros** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **1.0** | 34.0 | 113.0 | 4.0 | 171.0 | 3.0 | 18.0 | 13.0 | 34.0 | 7.0 | 11.0 | - | - | - |
| **2.0** | 19.0 | 331.0 | 14.0 | 545.0 | 7.0 | 4.0 | 24.0 | 17.0 | 2.0 | 5.0 | 1.0 | - | 1.0 |
| **3.0** | 1.0 | 22.0 | 17.0 | 366.0 | 25.0 | 27.0 | 54.0 | 27.0 | 5.0 | 1.0 | 3.0 | - | - |
| **4.0** | - | 2.0 | 10.0 | 107.0 | 12.0 | 62.0 | 50.0 | 56.0 | 11.0 | 21.0 | - | - | - |
| **5.0** | - | - | 3.0 | 4.0 | 5.0 | 119.0 | 4.0 | 95.0 | 18.0 | 22.0 | - | 3.0 | 1.0 |
| **6.0** | - | - | - | - | - | 20.0 | 1.0 | 9.0 | 5.0 | 33.0 | - | 1.0 | 7.0 |
| **7.0** | - | - | - | - | - | 3.0 | - | 2.0 | - | 19.0 | - | - | 2.0 |

Pelo *pairplot* abaixo, construído por meio da biblioteca Seaborn[[19]](#footnote-19), podemos perceber como se relacionam algumas variáveis.

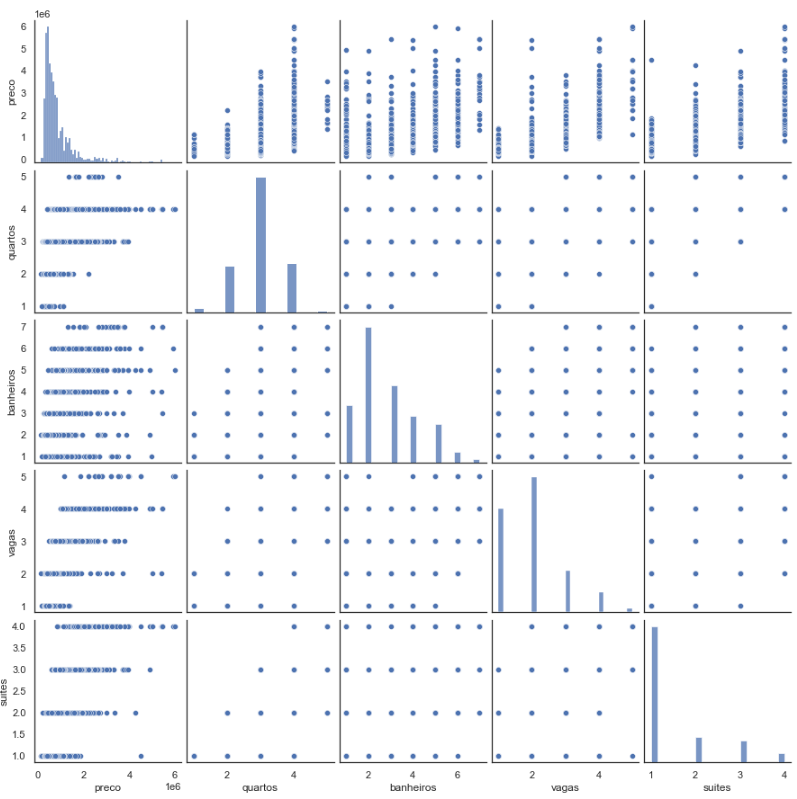
Figura - Pairplot - relação entre as variáveis



A mais nítida relação, como já mencionado anteriormente, é entre preço e área. Quanto maior a área, maior o preço.

Com relação a quartos, banheiros, vagas e suítes, percebemos que há uma relação direta no preço em sua base, ou seja, há aparentemente um piso a ser respeitado. Vamos ampliar a imagem para melhor visualizar

Figura - Comparativo preço x quartos, banheiros, vagas e suítes



Segue o código para construção dos pairplots.



# 4.1 Análise da correlação entre atributos

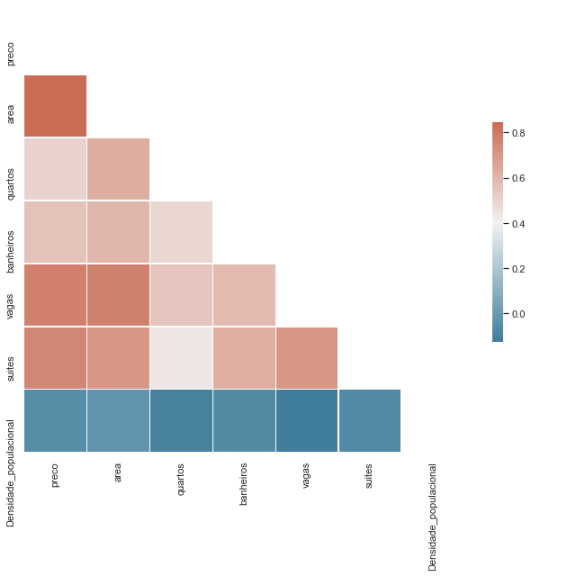
Um dos fatores mais importantes para o cálculo de regressão linear é a análise da correlação entre atributos.

Quando há variáveis correlacionadas, ou seja, quando uma variável aumenta de valor a outra também aumenta, ou, quando uma aumenta de valor a outra diminui, pode ocorrer de o modelo linear não ajustar seus pesos adequadamente, a depender da força da correlação.

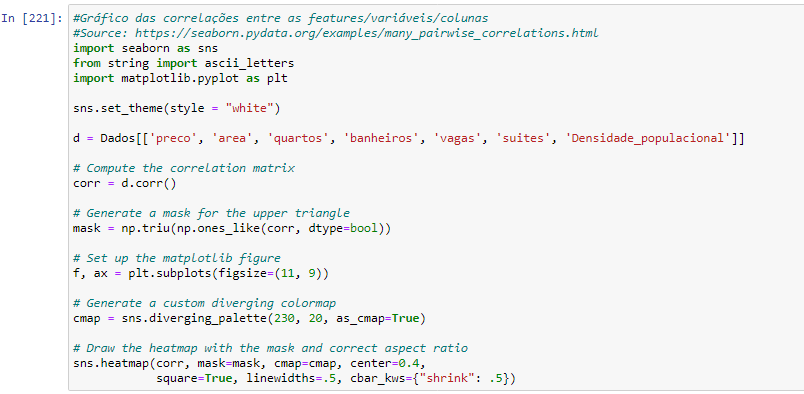
Nesses casos o modelo pode não servir para predizer resultados futuros, ou, no caso em análise, para fazer avaliação de imóveis de acordo com as ofertas no mercado.

Uma ferramenta muito utilizada para visualizar a correlação dos atributos é o mapa de calor abaixo[[20]](#footnote-20) o qual foi obtido por meio da biblioteca Seaborn e adaptado para o nosso caso.

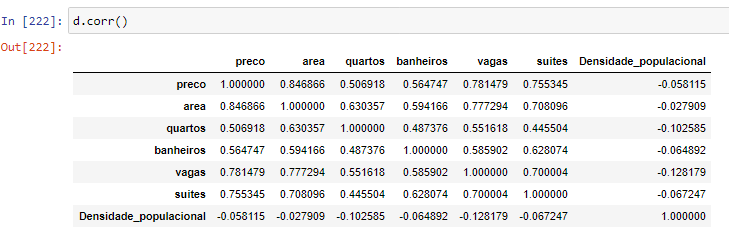
Figura - Heatmap - correlação dos atributos



Quanto mais próximo de 1 ou de -1, mais forte é a correlação dos atributos. No caso as variáveis mais correlacionadas são preço e área, preço e suítes, vagas e área, suítes e vagas. Abaixo segue o código para a geração do gráfico:



A tabela abaixo apresenta os valores de correlação dos atributos. Estima-se que as variáveis mais correlacionadas com a variável preço possam servir melhor para explicar o modelo.



# 4.2 Problema da dupla contagem de quartos e suítes

Comumente, em casos envolvendo imóveis, tem-se observado um problema de dupla contagem de quartos e suítes, bem como de banheiros e suítes.

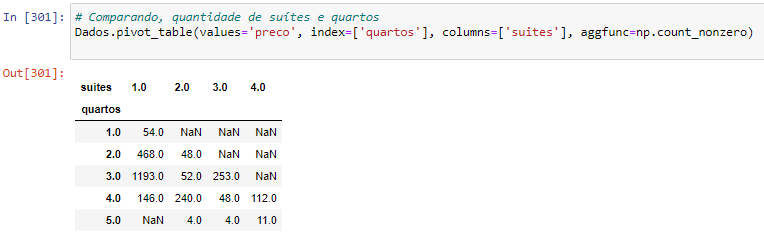
Este problema, em tese, tenderia a prejudicar o desempenho dos modelos de previsão ao alocar de parte do valor do imóvel ao quarto e parte à suíte, sendo que muitas vezes pode se tratar do mesmo espaço físico.

Ao tentar superar esse problema, vamos experimentar uma alternativa que considera um conceito diferente para quarto. Nessa abordagem quarto é conceituado como um quarto que não é uma suíte. Assim quarto será definido como um quarto sem banheiro e suíte como um quarto com banheiro.

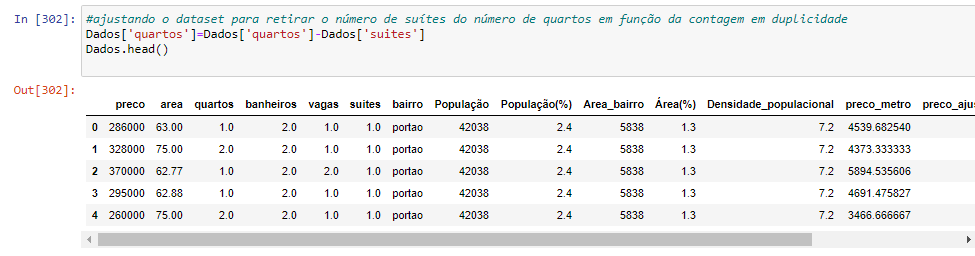
Definido o novo conceito de quarto, agora precisamos ajustar os valores da variável “quartos”. Vamos considerar que a quantidade de suítes está duplamente contada na variável “quartos”.

Desse modo procederemos à subtração da variável “quartos” pela quantidade de suítes, obtendo a quantidade de quartos sem banheiros.

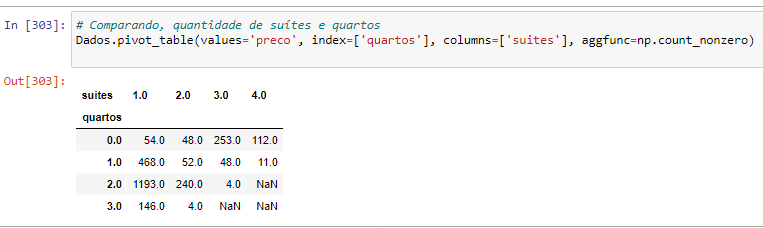
Abaixo vemos um resumo da quantidade de quartos comparado com suítes:



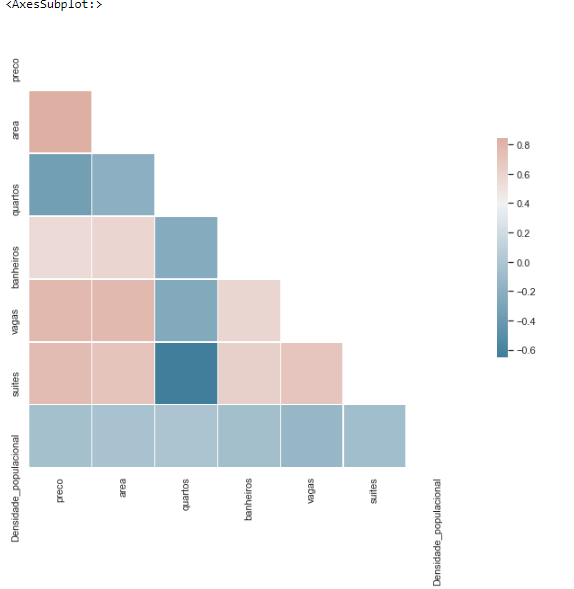
Na sequência fazemos a subtração mencionada acima, obtendo a quantidade de quartos sem suítes.



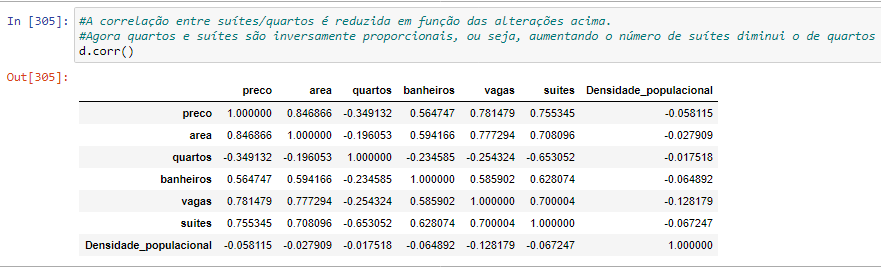
E obtendo novamente o resumo:



Observando novamente o mapa de calor vemos que as correlações entre preço e quarto, e, entre quarto e suítes, foram afetadas.



Vendo os números da tabela abaixo, percebemos que agora quartos e suítes são inversamente correlacionadas, o que nos parece bem lógico, pois nos apartamentos maiores há mais suítes e menos quartos sem banheiro.



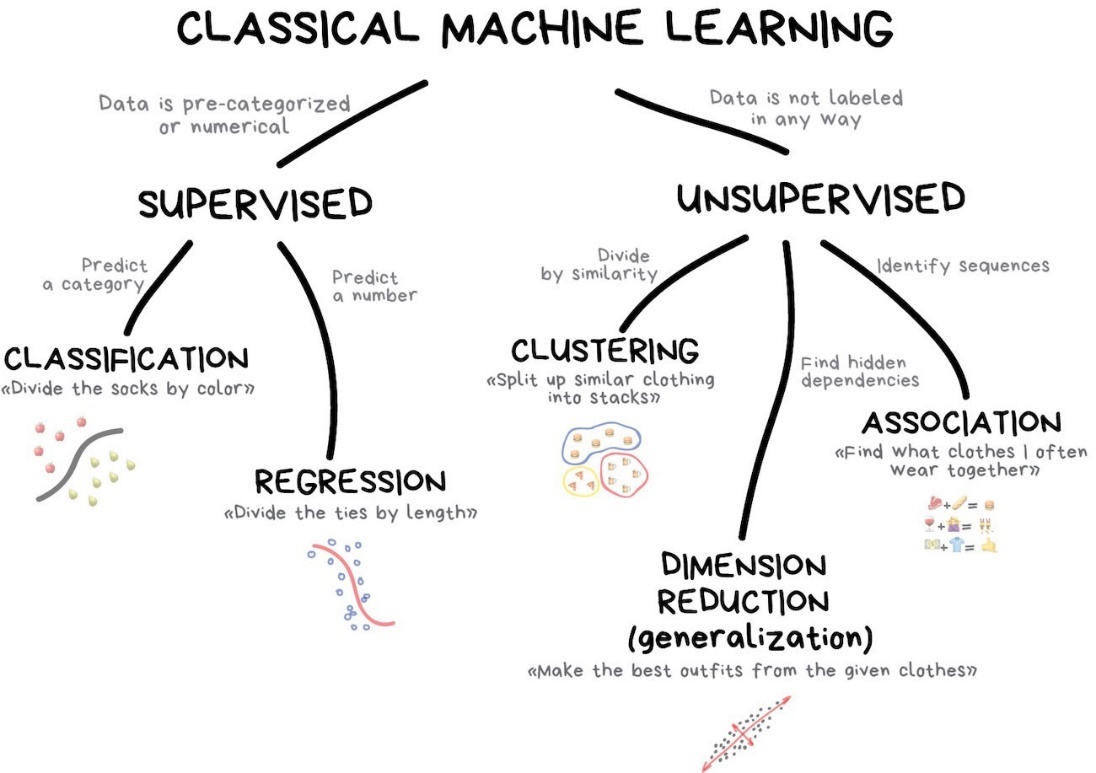
Com relação a banheiros e suítes não foi possível utilizar a mesma técnica, pois ao realizar a subtração de suítes do número de banheiros ocorreram vários registros com número negativo de banheiros. Acreditamos que seja decorrente da falta de critérios padronizados para anúncios, ou seja, enquanto alguns anunciantes incluem os banheiros das suítes no número de banheiros, outros anunciantes informam apenas os banheiros que não estão nas suítes.

Dessa forma, de modo a evitar a distorção dos dados, manteremos os registros de banheiros como estão no original.

# 5. Criação de Modelos de Machine Learning

Passaremos agora para a fase de modelagem e seguindo o mapa abaixo[[21]](#footnote-21), vemos que se trata de um problema de regressão linear pois os dados são pre-categorizados ou numéricos e almejamos prever um número, qual seja o preço dos apartamentos.

Figura - Mapa Classical Machine Learning



Os algoritmos de regressão linear que vamos utilizar estão disponíveis nas bibliotecas Statsmodel e Scikit-learn e são os seguintes:

1. Ordinary Least Squares (Statsmodel);
2. Ordinary Least Squares (Scikit-learn);
3. Regularized Regression Methods - Ridge Regression (Scikit-learn);
4. Least Absolute Shrinkage And Selection Operator regularization - LASSO (Scikit-learn);
5. Elastic Net (Scikit-learn);
6. Support Vector Regression - SVM-LinearSVR (Scikit-learn); e
7. Stochastic Gradient Descent: Regressor - SGDRegressor (Scikit-learn)

Embora contenha o mesmo algoritmo, os itens 1) e 2), além de se referir a bibliotecas diferentes, também tem o diferencial de que o algoritmo no item 2) será utilizado após a divisão dos dados em dados de treinamento e teste. Por esse motivo vamos utilizá-lo duas vezes.

Então vamos a apresentação dos detalhes de cada um deles.

# 5.1 Ordinary Least Squares (Statsmodel)

O que há de diferente nesse modelo dos modelos preliminares é que agora os dados já estão tratados e com o problema de dupla contagem resolvido.

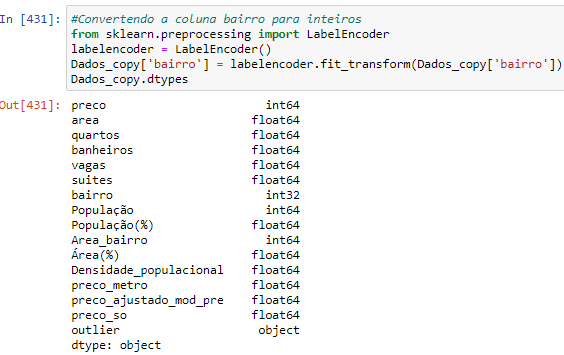
Vamos inicialmente fazer uma cópia do dataset, a qual chamaremos de “Dados\_copy” e, em seguida alterar o tipo de dado da variável “bairro” para “inteiros” por meio do algoritmo “LabelEncoder” da biblioteca Sklearn. A alteração do tipo de variável é devida para que o algoritmo de regressão produza um único coeficiente para essa variável

Em testes anteriores com a variável “bairro” do tipo category tínhamos um coeficiente para cada nome de bairro e, desse modo, ficava difícil excluir os itens com “Valor P” maior do que 0,05. Seguem os códigos:

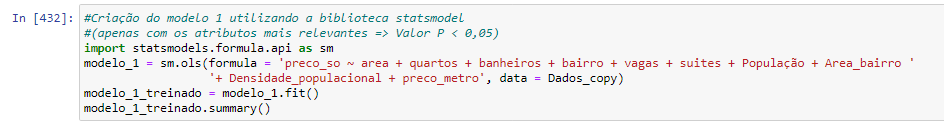
Figura - Visualização de colunas, criação da cópia e visualização dos tipos de variáveis



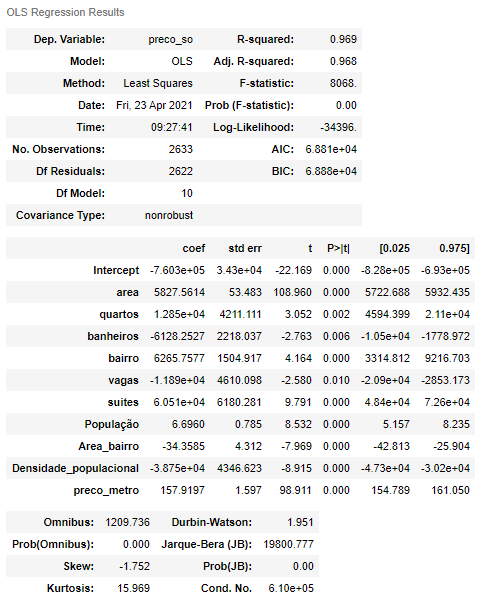
Figura - Transformação dos dados da variável "bairro"



Abaixo a criação e treinamento do modelo 1, utilizando como variável target a coluna “preco\_so” que contém os dados de preços de apartamento sem os outliers. Também, temos a invocação do summary com o resultado do treinamento (ajuste).



Na sequência temos o resultado do treinamento/ajuste do modelo 1.



Interessante observar uma pequena melhoria do Coeficiente de Determinação (R²) de 0.968 para 0.969. Os critérios de informação AIC[[22]](#footnote-22) (Akaike Information Criterion) e BIC (Bayesian Information Criterion[[23]](#footnote-23)) também tiveram uma pequena melhora de 6.884e+04 para 6.881e-04, e, de 6.889e+04 para 6.888e+04, respectivamente.

Embora seja comum utilizar os critérios de informação para comparação de modelos, vamos nos concentrar para efeito de comparação no Coeficiente de Determinação (R²).

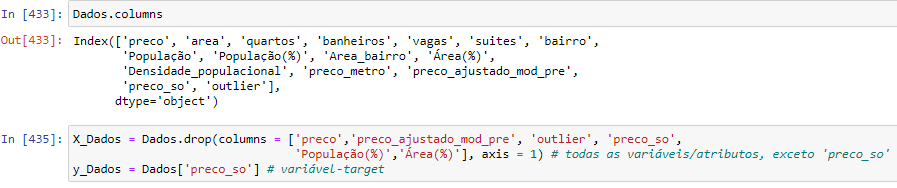
O diferencial entre esses dois modelos foi a inclusão das variáveis “quartos”, após o tratamento da dupla contagem, e a variável “bairro” após a transformação dos dados para o tipo “inteiro”.

# 5.2 Ordinary Least Squares (Scikit-learn)

Dessa vez vamos utilizar o mesmo algoritmo, só que fornecido pela biblioteca Scikit-learn.

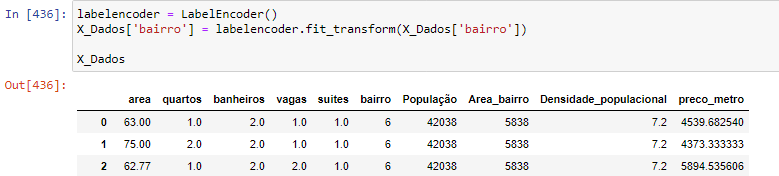
Vamos utilizar o dataset “Dados” e dividi-lo em variáveis preditoras (X\_Dados) e variável target (y\_Dados). Na construção de X\_Dados teremos o cuidado para remover as variáveis que não tem capacidade preditora (P value >0,05), já verificado em etapas anteriores, e igualando as features com o dataset “Dados\_copy” usado no modelo 1. Seguem os códigos:

Figura - Divisão do dataset em variáveis preditoras e variável target



Também faremos aqui o tratamento da variável “bairro” utilizando a ferramenta LabelEncoder.

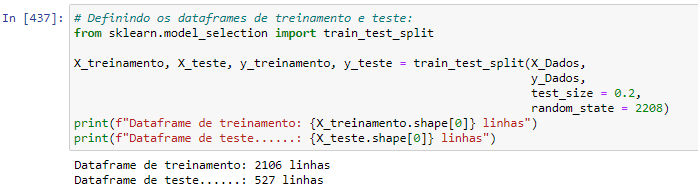
Figura - Transformação da variável "bairro" para o tipo "inteiro"



Na sequência, vamos fazer a divisão dos conjuntos de dados X\_Dados e y\_Dados em dados de treinamento e dados de testes por meio da ferramenta “train\_test\_split” da biblioteca Sklearn. Após essa divisão temos 2.106 registros para treinamento e 527 para teste.

Essa divisão se faz necessária para podermos avaliar o modelo com dados por ele desconhecidos. Assim evitamos problemas como “overfitting”, ou seja, o super ajustamento do modelo aos dados de treinamento, acarretando uma redução na capacidade de generalização e previsão.

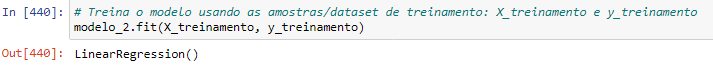
Figura - Divisão em dados de treinamento e teste



Continuando, já podemos importar a biblioteca com o algoritmo “LinearRegression” e instanciar o objeto na variável “modelo\_2”. Não vamos definir parâmetros diferentes dos predefinidos (default).

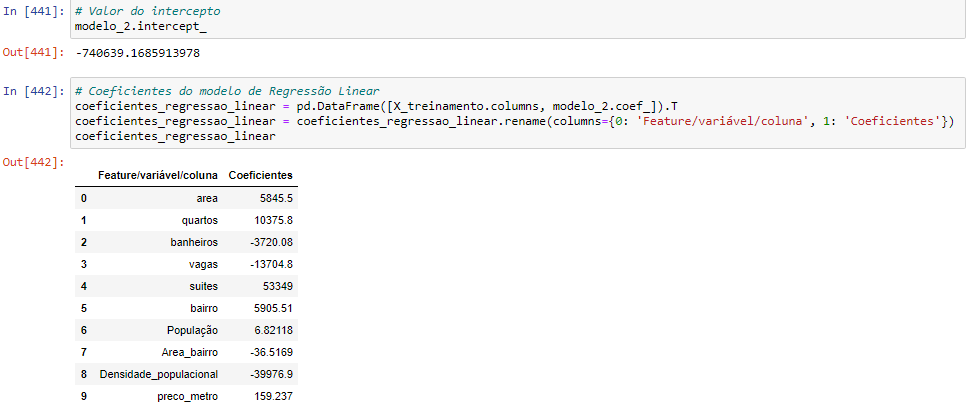


Faremos o ajuste do modelo com os dados de treinamento (X\_treinamento e y\_treinamento).



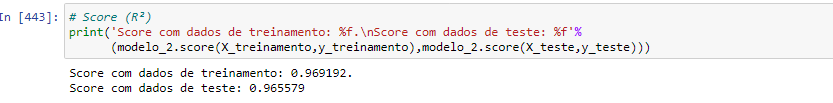
Podemos verificar abaixo o valor do intercepto e dos coeficientes obtido pelo modelo 2.

Figura - Valor do Intercepto e coeficientes



E agora, temos os valores do Coeficiente de Determinação (R²) calculado com os dados de treinamento e, também, com os dados de teste utilizando o modelo 2.

Figura - Coeficiente de Determinação (R²)



Verifica-se que, com os dados de teste há uma pequena piora do R², o que é normal de se esperar. Entretanto, entendemos que essa redução no R² é pequena, o que demonstra a alta capacidade de generalização do modelo.

Comparando com o modelo 1, o qual tem o mesmo valor de R², vemos que há pouco espaço para melhora, o que tentaremos a seguir por meio de outros algoritmos de regressão linear.

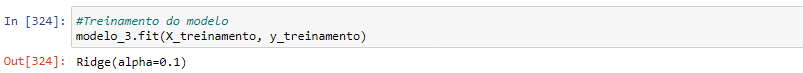
# 5.3 Regularized Regression Methods - Ridge Regression (Scikit-learn)

O algoritmo Ridge Regression é utilizado para resolver um dos problemas dos algoritmos anteriores (OLS), pois ele impõe uma penalidade no tamanho dos coeficientes, fazendo com que atributos correlacionados tenham coeficientes parecidos[[24]](#footnote-24).

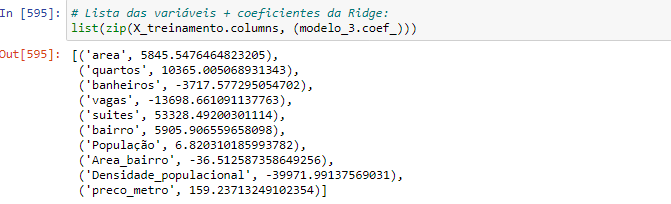
Vamos primeiro trabalhar em um modelo mais simples, importando o algoritmo Ridge da biblioteca Sklearn e instanciando o objeto. Como parâmetro definiremos alpha igual a 0,1.



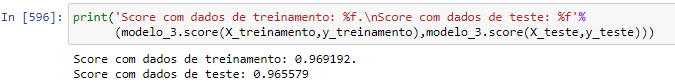
Treinamos o modelo com o código abaixo.



E verificamos os coeficientes.



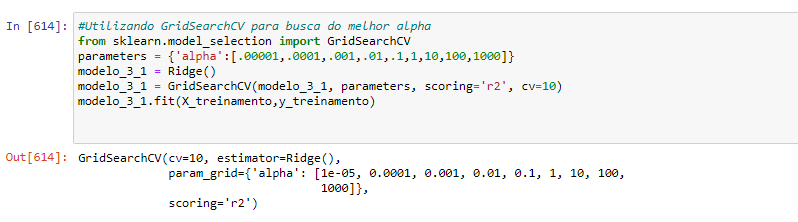
Por último verificamos o Coeficiente de Determinação (R²) e percebemos que não houve melhora.



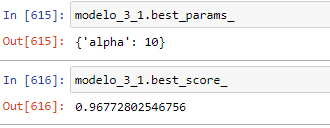
Agora vamos tentar buscar o valor de alpha que maximiza o resultado, utilizando a ferramenta GridSearchCV da biblioteca Sklearn.

Como parâmetros usaremos uma lista de nove valores para alpha que parte de 0,00001 e vai até 1.000, o algoritmo será Ridge, a métrica para score será R², e definiremos 10 partições para validação cruzada, conforme abaixo.

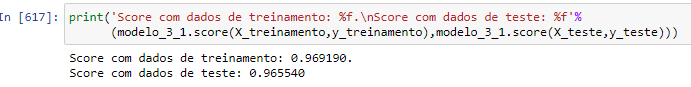
Figura - Busca do melhor alpha



Como resultado do modelo 3.1 temos como o melhor alpha sendo igual a 10 e o melhor score obtido na validação cruzada igual a 0,967728.



Ao avaliar o modelo com dados de treinamento e teste temos os seguintes Coeficientes de Determinação (R²).



Percebe-se que houve uma pequena redução do score com dados de treinamento, provavelmente decorrente da aplicação da penalização.

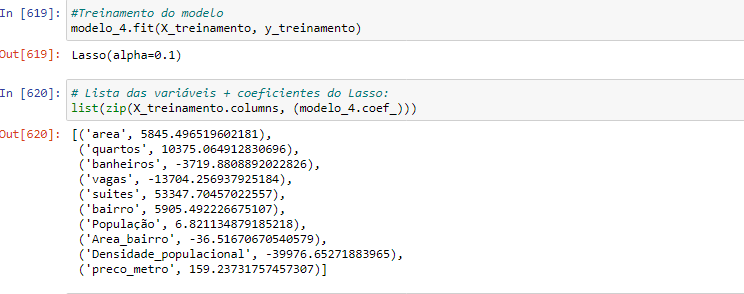
# 5.4 Least Absolute Shrinkage And Selection Operator regularization - LASSO (Scikit-learn)

Nesse tópico vamos abordar outro algoritmo que busca penalizar os coeficientes com o objetivo de diminuir a variância do modelo. Quando há múltiplas features altamente correlacionadas (ou seja, features que se comportam da mesma maneira) a regularização Lasso seleciona apenas uma dessas features e zera os coeficientes das outras, de forma a minimizar a penalização L1[[25]](#footnote-25).

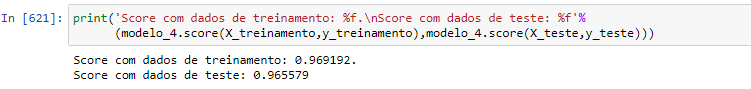
Da mesma forma que no tópico anterior, vamos inicialmente gerar um modelo maios simples. Importaremos o algoritmo LASSO da biblioteca Sklearn, instanciaremos o objeto na variável “modelo\_4” e definiremos como parâmetro o valor de alpha igual a 0,1, conforme abaixo.



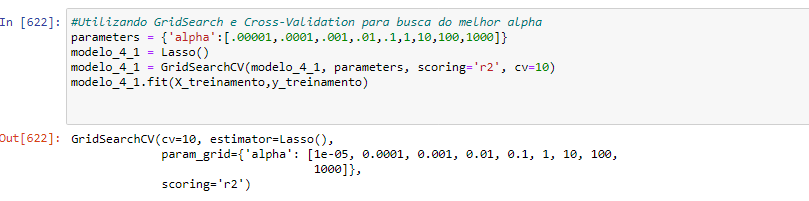
Em seguida vamos ajustar o modelo aos dados de treinamento e verificar os coeficientes.



Próximo passo é verificar os scores e mais uma vez não vemos melhoria dos Coeficientes de Determinação (R²).



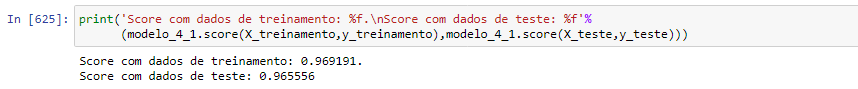
Novamente, tentaremos obter o melhor valor de alpha utilizando a ferramenta GridSearchCV. Definiremos os mesmos parâmetros do tópico anterior com exceção do *estimator* que será LASSO, quais sejam: uma lista de nove valores para alpha que parte de 0,00001 e vai até 1.000, a métrica para score será R², e 10 partições para validação cruzada. Efetuamos o ajuste do modelo conforme abaixo.



Com resultado temos o melhor parâmetro para alpha igual a 100 e melhor score igual a 0,967722.



Após fazer a validação nos dados de testes temos o seguinte:

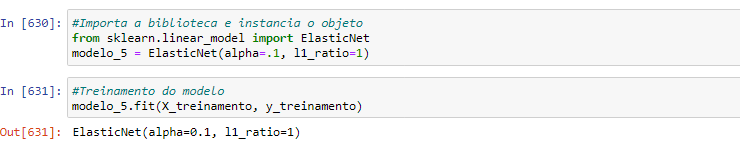


Mais uma vez verificamos que há uma pequena redução no R² com dados de treinamento e testes em consequência da penalização imposta pelo algoritmo.

# 5.5 Elastic Net (Scikit-learn)

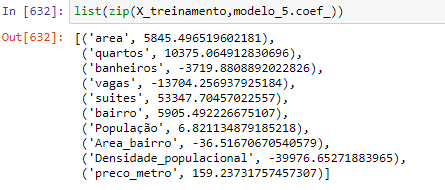
Agora vamos utilizar um algoritmo que combina os métodos de regularização apresentados nos últimos dois tópicos[[26]](#footnote-26). Também iniciaremos com um modelo mais simples e posteriormente tentaremos maximizar o R².

Primeiro importaremos o algoritmo ElasticNet da biblioteca Sklearn, instanciaremos o objeto na variável “modelo\_5” e definiremos os parâmetros de alpha igual a 0,1 e l1\_ratio igual a 1. Na sequência faremos o ajuste do modelo, conforme códigos abaixo.

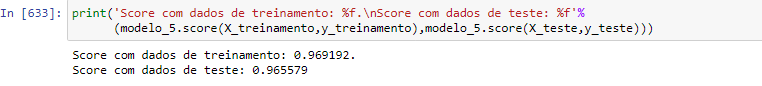


Abaixo podemos conferir os valores dos coeficientes.

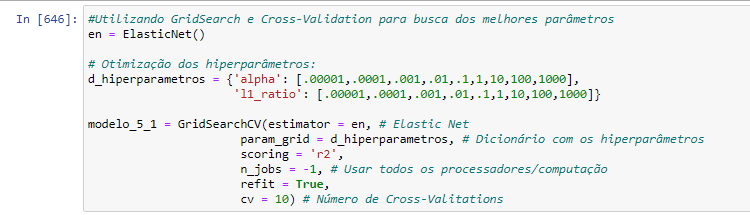
Figura - Coeficientes do modelo\_5



E por fim verificamos os scores (R²) com dados de treinamento e teste.

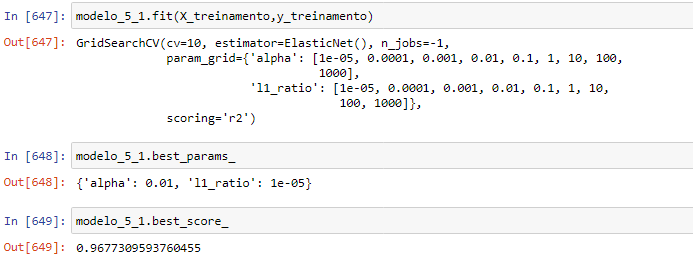


Aparentemente chegamos no limite, pois não conseguimos melhorar os scores. Vamos tentar novamente buscar os melhores parâmetro para o modelo utilizando as ferramentas GridSearchCV. Como parâmetros definiremos uma lista de nove valores para alpha e l1\_ratio que parte de 0,00001 e vai até 1.000, a métrica para score será R², 10 partições para validação cruzada, número de Jobs igual a -1, significando que serão utilizados todos os processadores, *refit* igual a *True* significando que será reajustado o *estimator* usando os melhores do *dataset* inteiro. O *estimator*, obviamente será o ElasticNet. Vejamos o código.

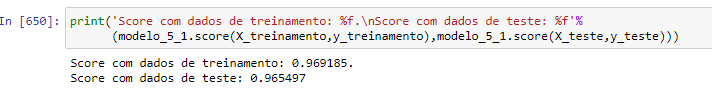


Treinaremos o modelo e conferimos os melhores parâmetros e scores.

Figura - Treinamento do modelo 5.1



Agora vamos verificar os scores (R²) nos dados de treinamento e de teste.



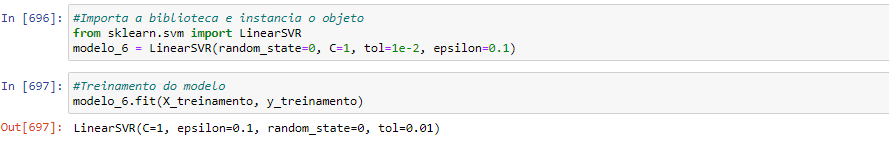
Mais uma vez vemos uma pequena redução nos scores em função das penalizações impostas pelo algoritmo.

# 5.6 Linear Support Vector Regression - SVM-LinearSVR (Scikit-learn)

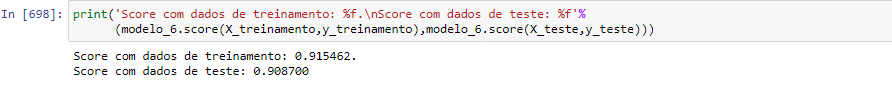
Dessa vez utilizaremos o algoritmo Linear Support Vector Regression, o qual se trata de uma versão especializada do algoritmo Support Vector Machine para problemas de regressão linear, pois utiliza apenas o kernel linear. Assim, o algoritmo se torna mais leve e eficiente para escalar mais dados[[27]](#footnote-27).

Seguindo a ordem natural vamos importar a biblioteca e instanciar o objeto. Como parâmetros definiremos, nesse primeiro modelo, C=1 (força do parâmetro de regularização), tol=1e-2 (tolerância para critério de stop), e épsilon=0.1(função de perda).

Figura - Criação a ajuste do modelo 6 - SVR (sklearn)

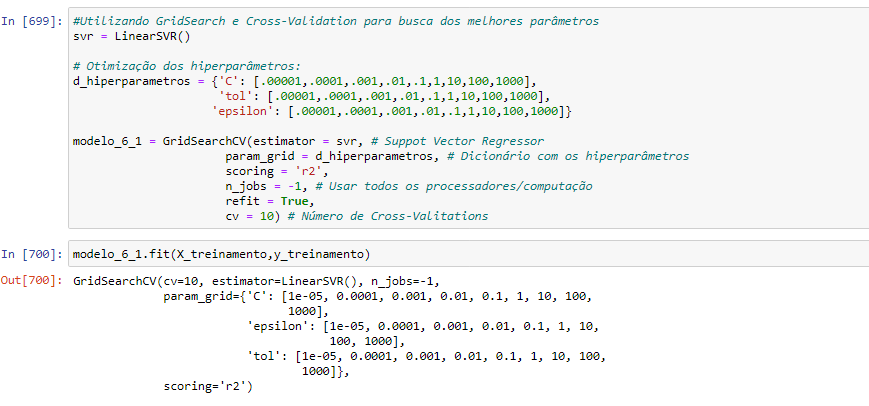


Como resultado temos um score muito aquém do obtido nos modelos anteriores.

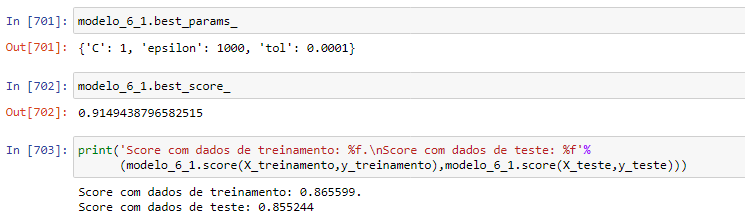


Na sequência, tentaremos obter os parâmetros ideais, utilizando a ferramenta GridSearchCV. Para seleção dos parâmetros “C”, “tol” e “épsilon”, apresentamos uma lista de múltiplos de 10 que parte de 0,00001 e vai até 1.000. Definimos como critério de “scoring” o R², utilizar todos os processadores como o “n\_jobs” igual a -1, “refit” igual a “True” para utilizar os melhores parâmetros no dataset inteiro e, também, definimos 10 partições para o cross validation.

Figura - Criação do modelo 6.2 com GridSearchCV



Após o treinamento obtivemos como melhores parâmetros “C” igual a 1, épsilon igual a 1000 e tolerância igual a 0.0001, sendo que não houve uma melhora no score, mas, pelo contrário, houve uma piora significativa.



Deixando esse modelo de lado, partiremos para o próximo.

# 5.7 Stochastic Gradient Descent: Regressor - SGDRegressor (Scikit-learn)

Esse modelo linear, durante a descida do gradiente, calcula a função de perda (*loss function*) de forma estocástica, ou seja, a cada exemplo[[28]](#footnote-28).

Primeiro vamos fazer um modelo inicial usando parâmetros arbitrários. Nesse modelo vamos utilizar outras duas ferramentas disponível na biblioteca sklearn, a “make\_pipeline” e “StandardScaler”.

A ferramenta “make\_pipeline”[[29]](#footnote-29) tem por objetivo organizar um pipeline, ou seja, uma ordem de passos ou comandos a ser seguido pelo script. No caso queremos primeiro escalar os dados de forma padronizada, ou seja, transformá-los na escala padrão, para depois fazer o ajuste do modelo.

E é essa transformação dos dados que é realizada pela ferramenta “StandardScaler”[[30]](#footnote-30). Em suma, a ferramenta realizará o seguinte cálculo nos dados das variáveis preditoras:

z = (x - u) / s

onde:

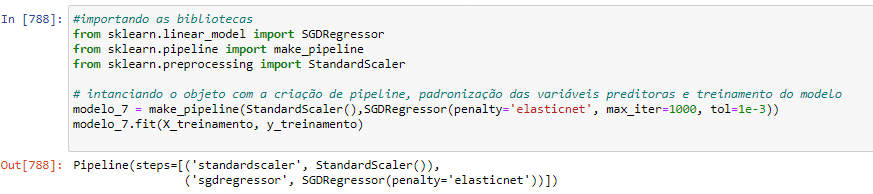
x é a variável a ser transformada;

u é a média dos exemplos (x);

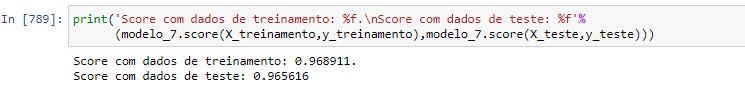
s é o desvio padrão dos exemplos (x)

Os parâmetros definidos para o modelo foram “penalty” igual a “elasticnet”, “max\_iter” igual a 1000 e “tol” igual a 0,001, conforme abaixo:

Figura - Criação do modelo 7 com o uso das ferramentas make\_pipeline, StandardScaler e SCDRegressor

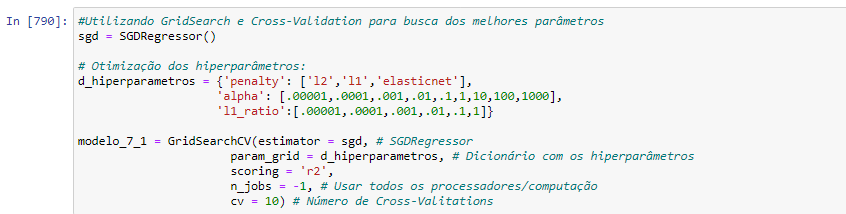


O resultado (R²) ficou uma pouco abaixo dos melhores modelos até agora.



Na sequência, vamos tentar obter os melhores parâmetros com o uso da ferramenta GridSearchCV. Definiremos os seguintes parâmetros no GridSearchCV: *estimator* igual a SGDRegressor; *scoring* igual a “r2”; n\_*jobs* igual a -1, indicando o uso de todos os processadores; cv igual a 10, ou seja, 10 partições para cross validation; e, no grid, como parâmetros do estimator, temos para “penalty” as *strings* “l2”, “l1” e “elasticnet”; para valores de “alpha” uma lista que vai de 0,00001 até 1.000; e para valores de “l1\_ratio” uma lista que vai de 0,00001 até 1. Segue o código.

Figura - Criação do modelo 7.1 com o uso de GridSearchCV e SGDRegressor

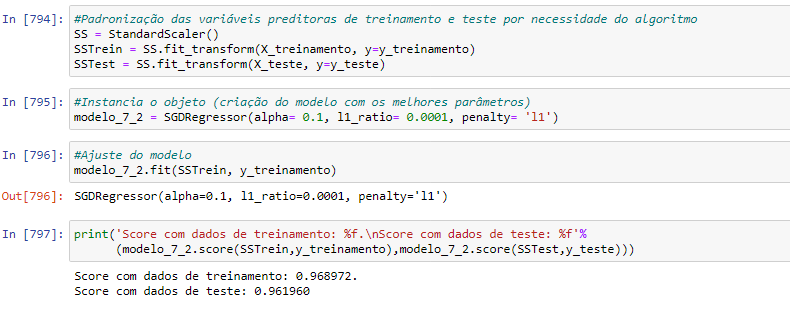


Como melhores parâmetros e melhor score temos o seguinte:



Agora vamos padronizar as variáveis preditoras e criar outro modelo apenas com os melhores parâmetros e visualizar o score. Seguem os códigos.

Figura - Criação do modelo 7.2 com melhores parâmetros



Vemos acima que o score (R²) ficou próximo aos modelos já treinados em tópicos anteriores, o que pode indicar que chegamos no limite de melhoria.

# 6. Apresentação dos Resultados

Nesta seção vamos apresentar os resultados obtidos por meio do modelo de Canvas proposto por Vasandani, por um comparativo entre os modelos e, também, por meio de um *dashboard* elaborado por meio da biblioteca Bokeh[[31]](#footnote-31).

# 6.1 Workflow da análise de dados

Segue abaixo um Canvas onde podemos visualizar o *workflow* do trabalho desenvolvido neste TCC.

|  |  |  |
| --- | --- | --- |
| **1. Definição do Problema (*Problem Statement*):**  Analisar os dados de preços de imóveis (apartamentos) de sete bairros da cidade de Curitiba-PR, bem como dados referentes ao perfil demográfico (densidade populacional) para fins de precificação dos imóveis. | **2. Resultados e previsões (*Outcomes/Predictions*):**  Estima-se que os atributos inerentes aos imóveis, como área, número de quartos etc., sejam preponderantes no seu preço. Buscaremos investigar o quanto dados geográficos referente à localização dos apartamentos como densidade populacional podem afetar o preço. | **3. Aquisição dos dados (*Data Acquisition*):**  Os dados foram obtidos de duas fontes distintas. Uma com os dados de anúncios de imóveis disponível em uma disciplina de estatística da UFPG e outra com dados geográficos oriundos do Censo de 2010 inseridas em um estudo do DIEESE de 2016. Apesar de serem de períodos diversos, o resultado da análise não será afetado. |
| **4. Modelagem (*Modeling*):**  Após o devido tratamento dos dados, utilizando -se de abordagens diferenciadas para tratamento de *outliers*, foi realizada a modelagem com diversos algoritmos de regressão linear disponíveis nas bibliotecas Statsmodel e Scikit-Learn. | **5. Avaliação do modelo (*Model Evaluation*):**  Os modelos foram avaliados utilizando como *score* o Coeficiente de Determinação (R²), sendo que em geral os resultados foram similares, indicando um grande ajustamento do modelo aos dados, tanto de treinamento quanto de teste. | **6. Preparação dos dados (*Data Preparation*):**  Os dados ausentes, duplicados e outliers foram devidamente tratados de acordo com a técnica pertinente. Por diversas vezes foi preferível excluir os registros para evitar a distorção dos dados. |

# 6.2 Comparativo dos modelos

Feita a apresentação inicial do workflow, vamos agora comparar os modelos, inicialmente verificando seus coeficientes na tabela abaixo que foi confeccionada pelo seguinte código:

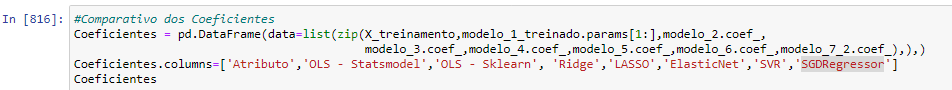
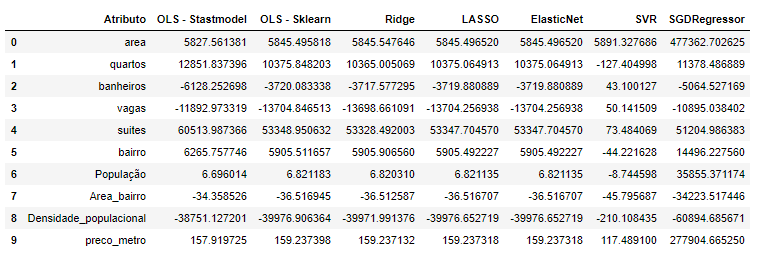


Figura - Coeficientes dos modelos



Podemos ver nos coeficientes que os modelos OLS – Sklearn, Ridge, LASSO e ElasticNet possuem pequenas divergências entre eles devidos aos parâmetros de regularização. Entretanto, podemos dizer que os coeficientes desses modelos possuem praticamente os mesmos valores.

Os coeficientes do modelo OLS – Statsmodel também estão próximos dos citados no parágrafo anterior, as divergências podem ser atribuídas ao procedimento de separação dos dados em dados de treinamento e dados de teste.

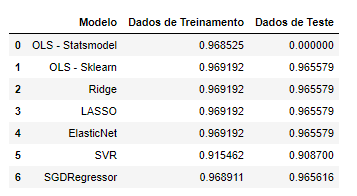
Já o modelo SVR apresentou o coeficiente de determinação (R²) muito abaixo dos demais, provavelmente por isso que seus coeficientes estão tão diferentes dos outros modelos.

E, por último, quanto ao modelo SGDRegressor podemos afirmar que os coeficientes estão considerando os dados padronizados, com a média em zero, sendo por isso que os coeficientes possuem valor tão elevado. Assim, não servem para efeito de comparação com os demais modelos.

Agora veremos como ficou o *score* dos modelos considerando o coeficiente de determinação (R²). Cabe registrar que foram selecionados os melhores *scores* de cada algoritmo nos dados de treinamento, independentemente de ser um modelo inicial ou um modelo mais bem trabalhado.

A tabela abaixo apresenta os *scores* por modelo:

Figura - Scores dos modelos (R²).



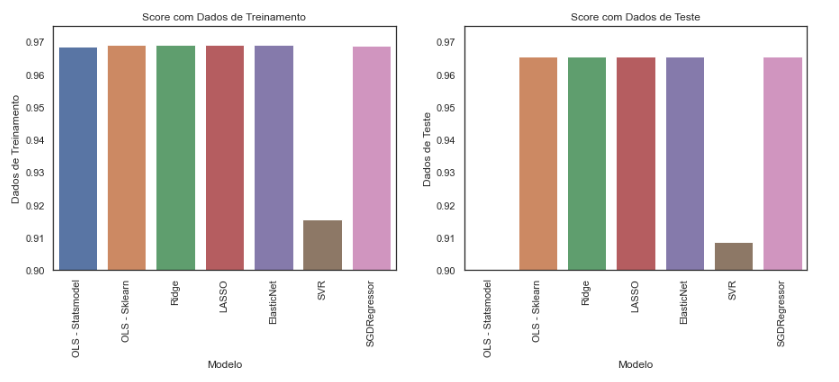
O modelo OLS – Statsmodel foi treinado com toda a base de dados, ou seja, não foi dividida a base em treinamento e teste. Por esse motivo que não há *score* para dados de teste.

Podemos ver que os modelos OLS – Sklearn, Ridge, LASSO e Elasticnet apresentaram o melhor desempenho com dados de treinamento. Já com os dados de teste o melhor modelo foi o SGDRegressor, embora tenha um resultado ligeiramente inferior aos anteriores nos dados de treinamento.

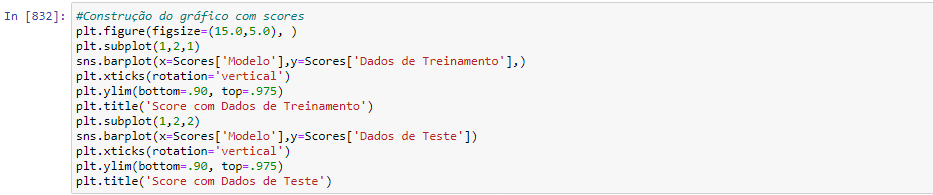
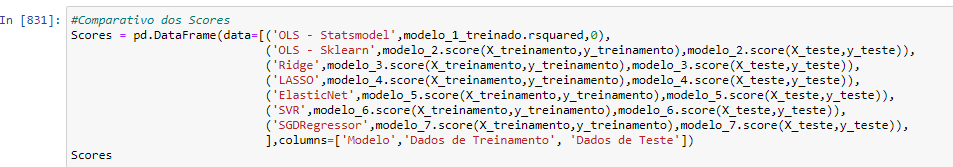
O pior modelo no quesito *score* foi o SVR, tanto nos dados de treinamento, quanto nos dados de teste. Estima-se que tal resultado inferior seja devido a uma maior dificuldade de escolha dos parâmetros ideais e não ao algoritmo em si.

O gráfico a seguir apresenta o mesmo resultado da tabela acima para fins de melhor visualização. Percebe-se que os *scores* estão, aparentemente, no limite de melhoria e que avanços adicionais talvez tenham que ser obtidos por meio de *features engineering*, ou seja, já na fase de coleta dos dados.

Figura - Scores dos modelos (R²)



Abaixo podemos ver os códigos para a construção da tabela e do gráfico com os *scores.*

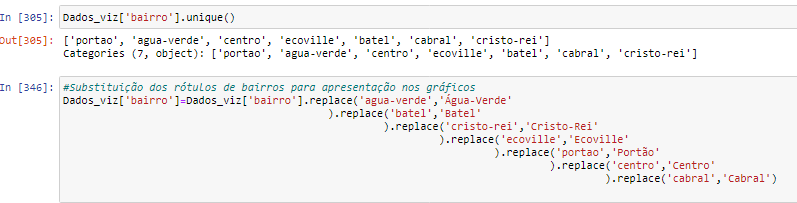


# 6.3 *Dashboard*

Por último vamos visualizar um *dashboard* desenvolvido por meio da biblioteca Bokeh[[32]](#footnote-32), o qual se trata de um arquivo no formato .html.

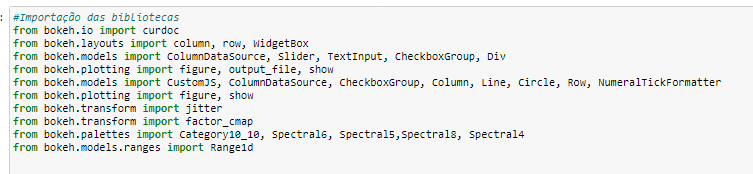
Inicialmente, para a elaboração dos gráficos, foram efetuadas algumas alterações no *dataframe* para melhor visualização, como mudança no nome dos bairros. Os códigos podem ser vistos abaixo:

Figura - Alterações no dataframe para melhor visualização



A seguir foram realizadas as importações da biblioteca Bokeh.

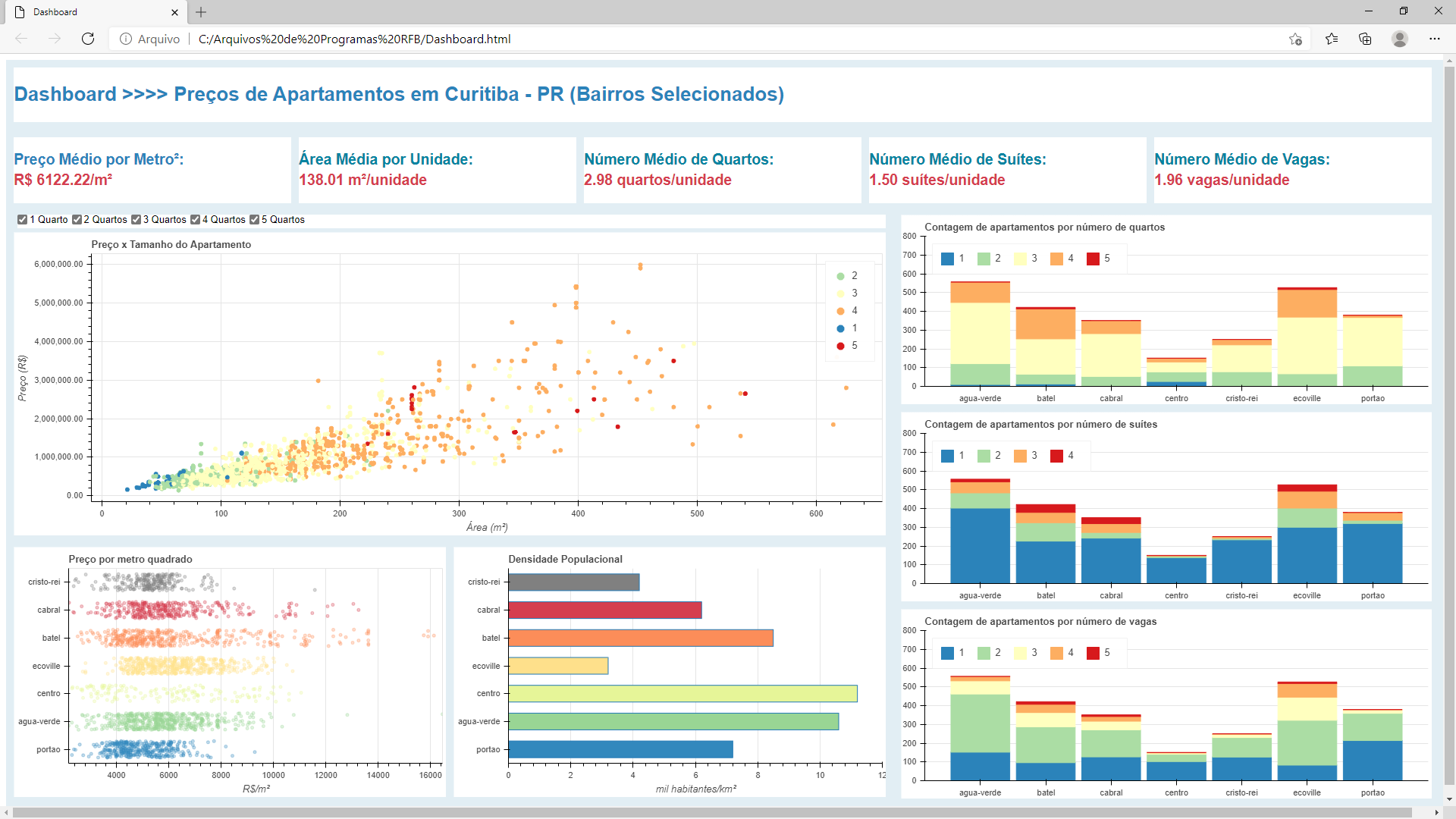
Figura - Importações de bibliotecas



Este *dashboard* possui cinco seções conforme abaixo:

1. Título (no topo);
2. Principais indicadores (logo abaixo do título);
3. Gráfico principal interativo, com dados de preço x área (centro);
4. Gráficos de barras horizontais (2), um com dados de preço/m² e outro com densidade populacional segregados por bairro (à esquerda e abaixo); e
5. Gráficos de barras verticais empilhadas (3), com dados de quartos, suítes e vagas de estacionamentos, segregados por bairros (abaixo e à direita).

Figura – Dashboard – mapa das seções



**5. Gráficos de barra verticais empilhadas**

**4. Gráficos de barras horizontais**

**3. Gráfico principal**

**2. Principais indicadores**

**1. Título**

A seguir teceremos comentários sobre cada seção.

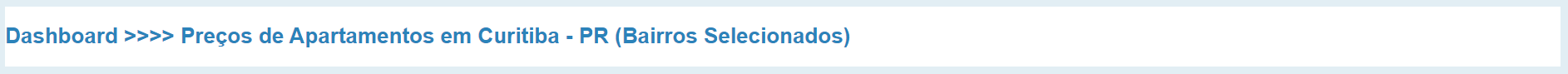
O título está localizado em uma “Div” na parte superior do *dashboard* e foi construído com o código abaixo. Parte deste código é composto por linguagem Python e parte por HTML na *string* passada no parâmetro *text da Div.*

Figura - Código para geração do título



Detalhe do título.

Figura - Título



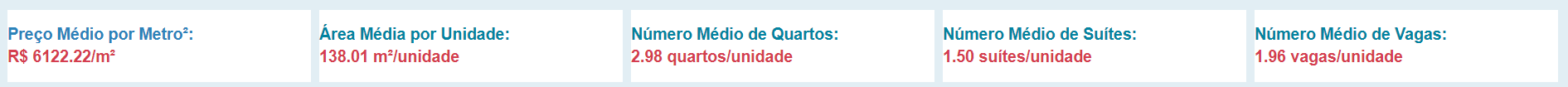
Os principais indicadores são em número de cinco e estão apresentados em outra *Div* logo abaixo do título. Contém informações gerais sobre a média dos principais componentes das unidades (apartamentos), considerando todos os registros do *dataset.* Da mesma forma que no título, contém fragmentos de ambas as linguagens de programação (python e html). Segue o código:

Figura - Código para geração dos principais indicadores



Detalhe dos principais indicadores.

Figura - Principais indicadores



O gráfico principal conta com um CheckboxGroup onde é permitido aos usuários selecionarem a quantidade de quartos que desejam visualizar no gráfico. Também conta com uma *hover* com informações numéricas sobre área e preço. As cores são definidas usando um *hue* com a paleta *Spectral5*, facilitando a distinção da quantidade de quartos por apartamento e harmonizando com o restante do *dashboard* que também foi construído usando tons do grupo de paletas *Spectral*.

O grande desafio na elaboração desse gráfico foi incluir no repertório mais uma linguagem de programação, a *Javascript,* pois a dinâmica do gráfico após a geração do arquivo html é comandada por essa linguagem no *backend* do *browser*. Então foi necessário incluir nos parâmetros do *CheckboxGroup* um *call-back* com o código *javascript* que operacionalize a dinâmica de alteração do gráfico após a seleção de quartos na caixa de seleção.

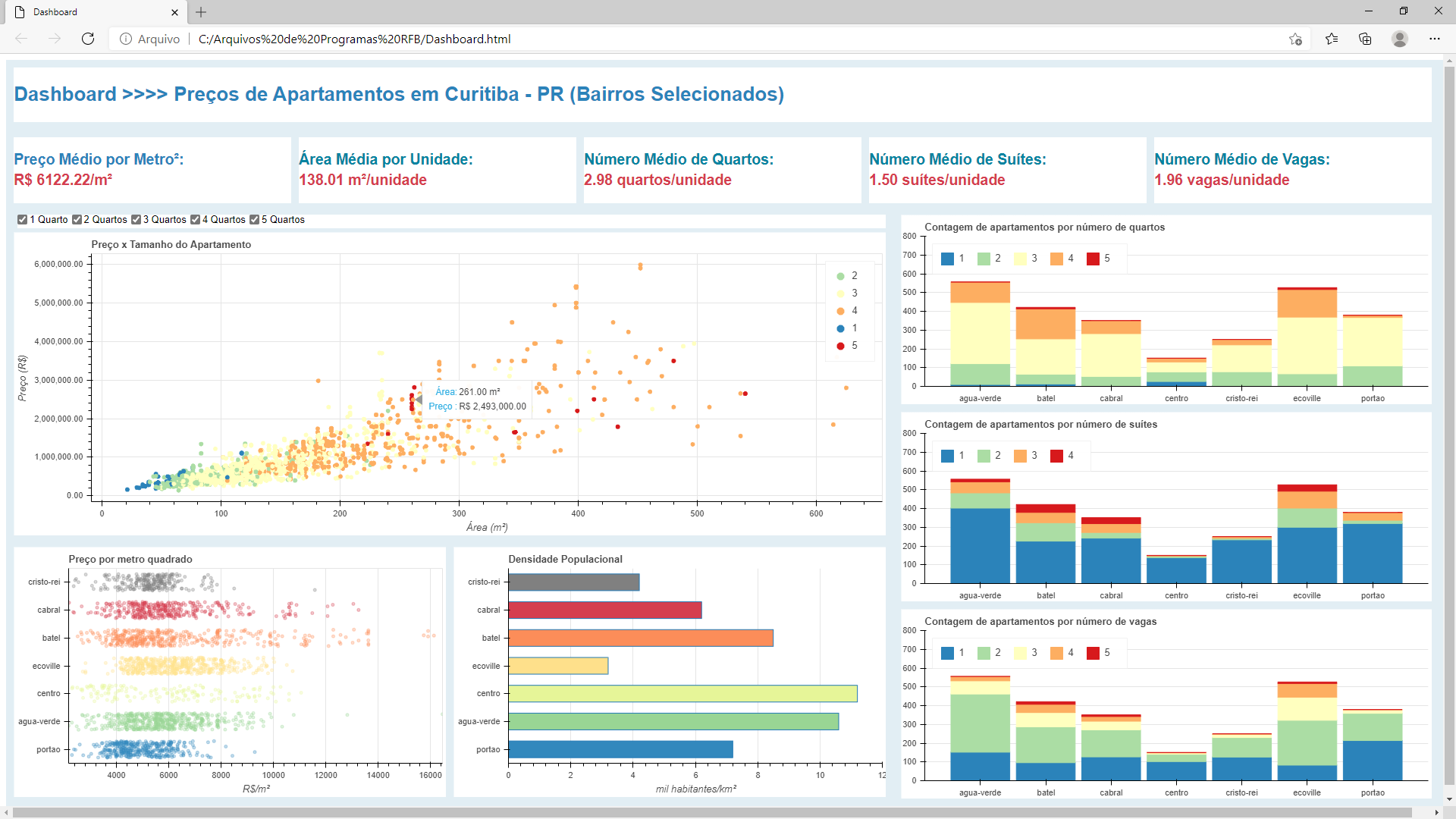
O código pode ser visualizado abaixo.

Figura - Código para criação do gráfico principal



Detalhe do gráfico principal.

Figura - Detalhe do gráfico principal com hover acionado

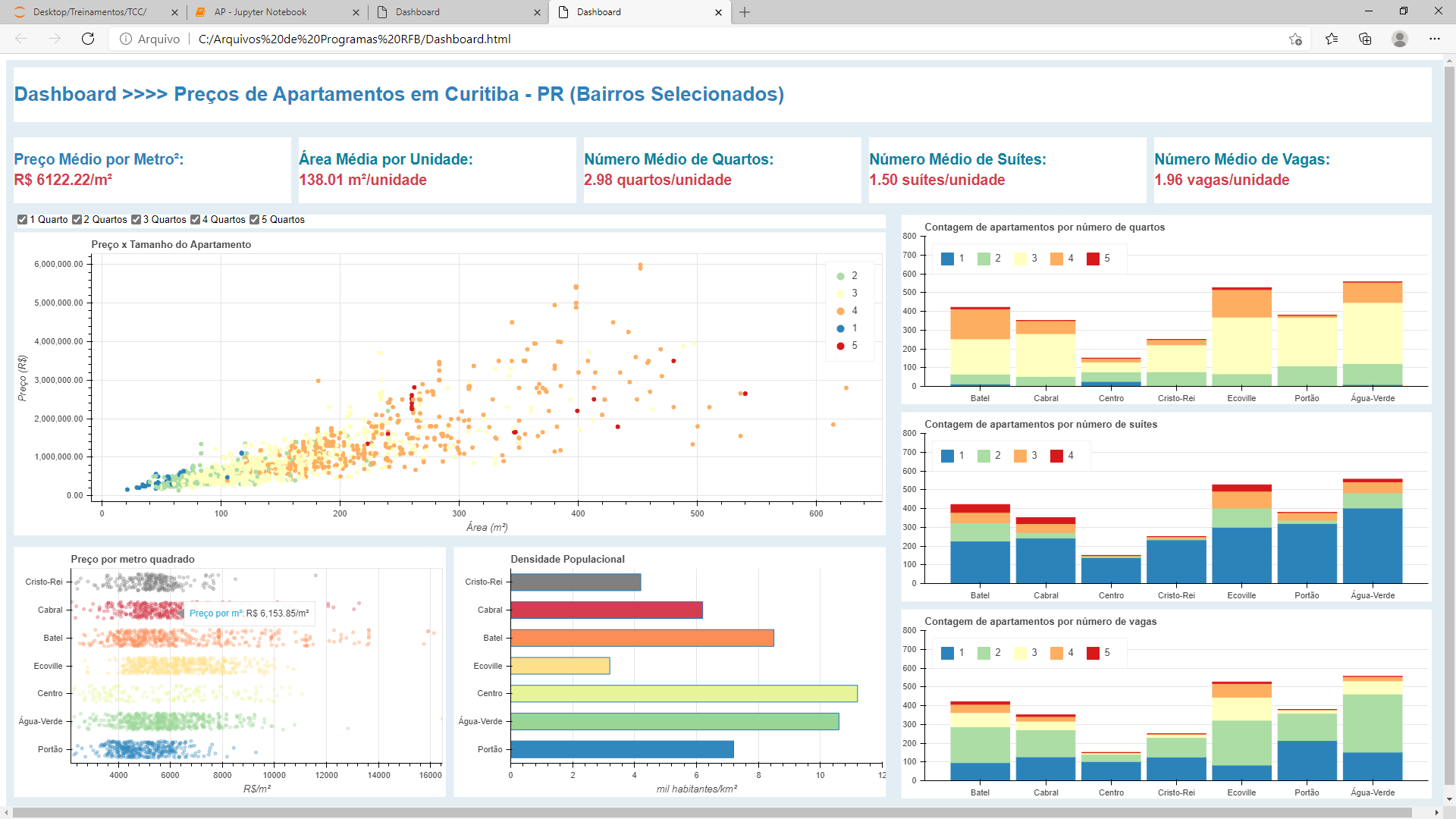


Os gráficos de barras horizontais foram apresentados com o intuito de fazer a comparação do preço do metro quadrado dos apartamentos com a densidade populacional dos bairros de Curitiba-PR. Nesse caso, o usuário pode verificar em qual faixa de preço/m² estão os apartamentos em cada bairro e ver se a concentração de pessoas naquele bairro é um fator que pode influenciar no preço. Também pode verificar em quais bairros há uma dispersão maior de valores, como no bairro Batel onde há variação no preço/m² que vai desde menos de R$ 4.000,00/m² até cerca de R$ 16.000,00/m². Contam com um *hover* com informações numéricas sobre os gráficos, dando maior interatividade com o usuário. Os códigos podem ser visualizados a seguir.

Figura - Código para criação dos gráficos de barras horizontais



Detalhe dos gráficos de barras horizontais com *hover acionado.*



Os gráficos de barras verticais empilhadas têm por finalidade apresentar outras grandes variáveis que impactam no valor dos imóveis, quais sejam quartos, suítes e vagas de estacionamento. Todos foram elaborados com paletas do grupo *Spectral* e apresentam no eixo “y” a contagem de apartamentos de acordo com a quantidade de itens (quartos, suítes, vagas). Também possuem um *hover* com informações numéricas como números de quartos (auxiliando a legenda) e quantidade de unidades (apartamentos). Para agregar mais informação no gráfico, foram segregadas as colunas por bairros para melhor visualização da distinção dos portes de apartamentos em cada um deles, ou seja, em quais bairros estão os maiores apartamento, com mais suítes, etc. O código em python segue abaixo:

Figura – Código para geração do gráfico de barras vertical empilhado 1 - quartos

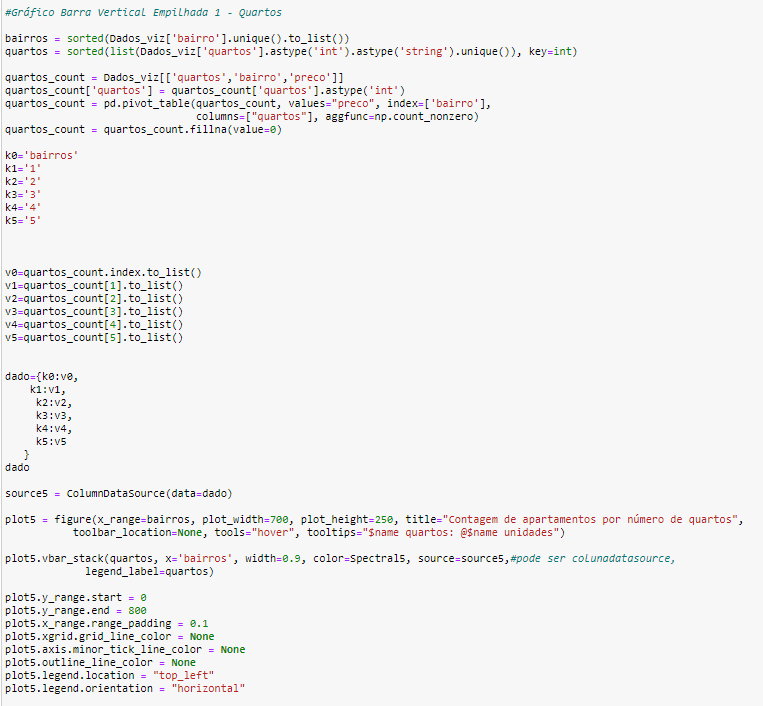


Figura - Código para geração do gráfico de barras vertical empilhado 2 - suítes

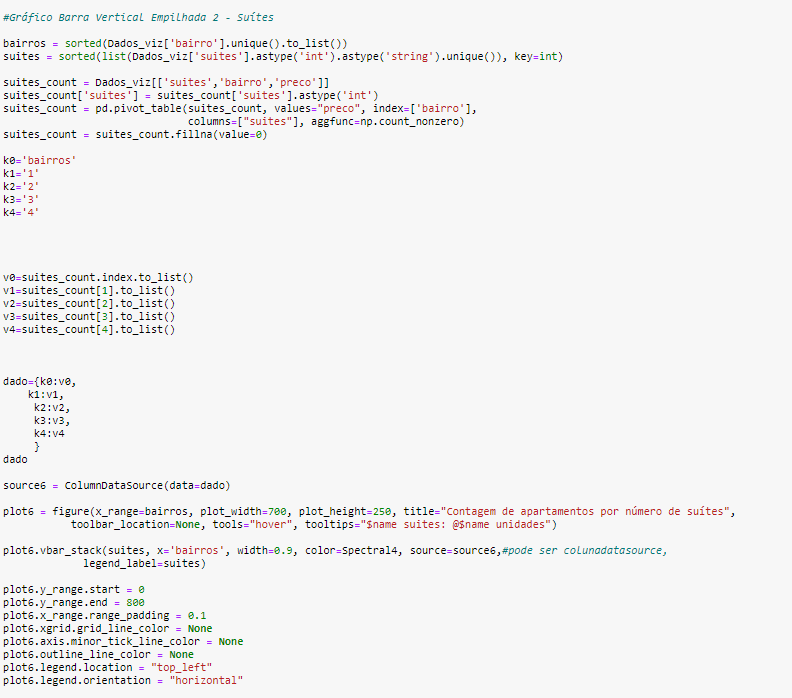
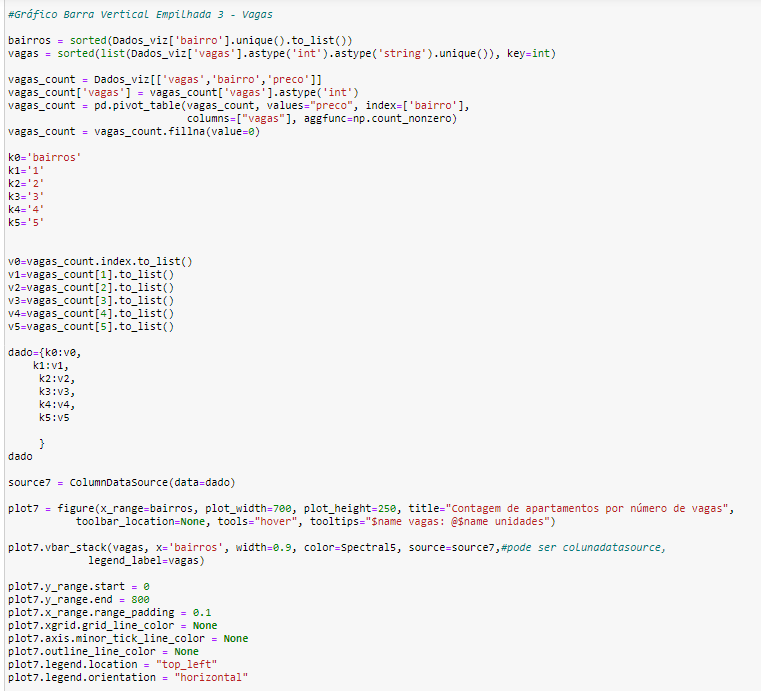
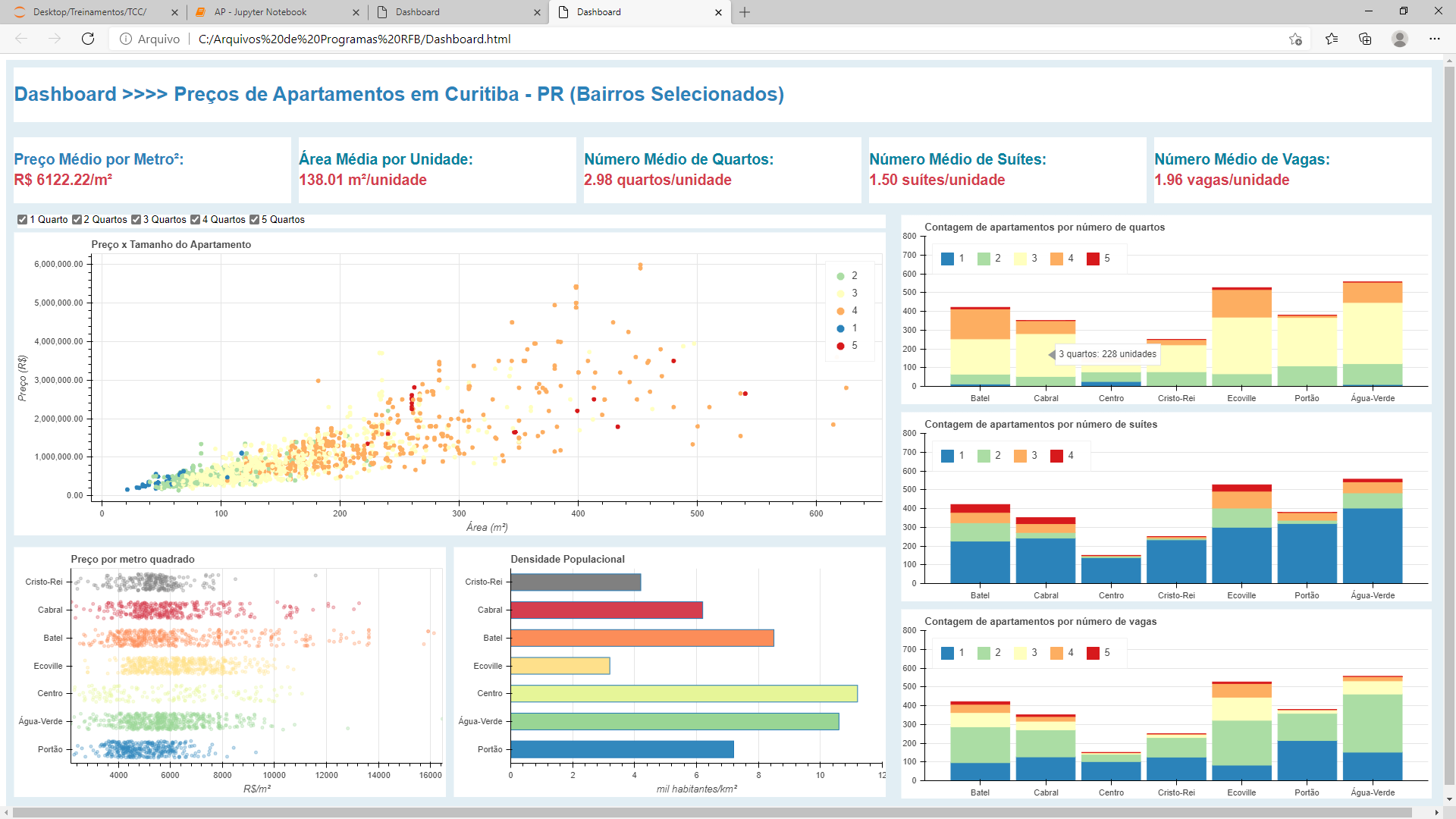


Figura - Código para criação do gráfico de barras vertical empilhado 3 - vagas de estacionamento



Detalhe dos gráficos de barras verticais empilhadas com *hover acionado.*

Figura - Gráficos de barras verticais empilhadas com hover acionado



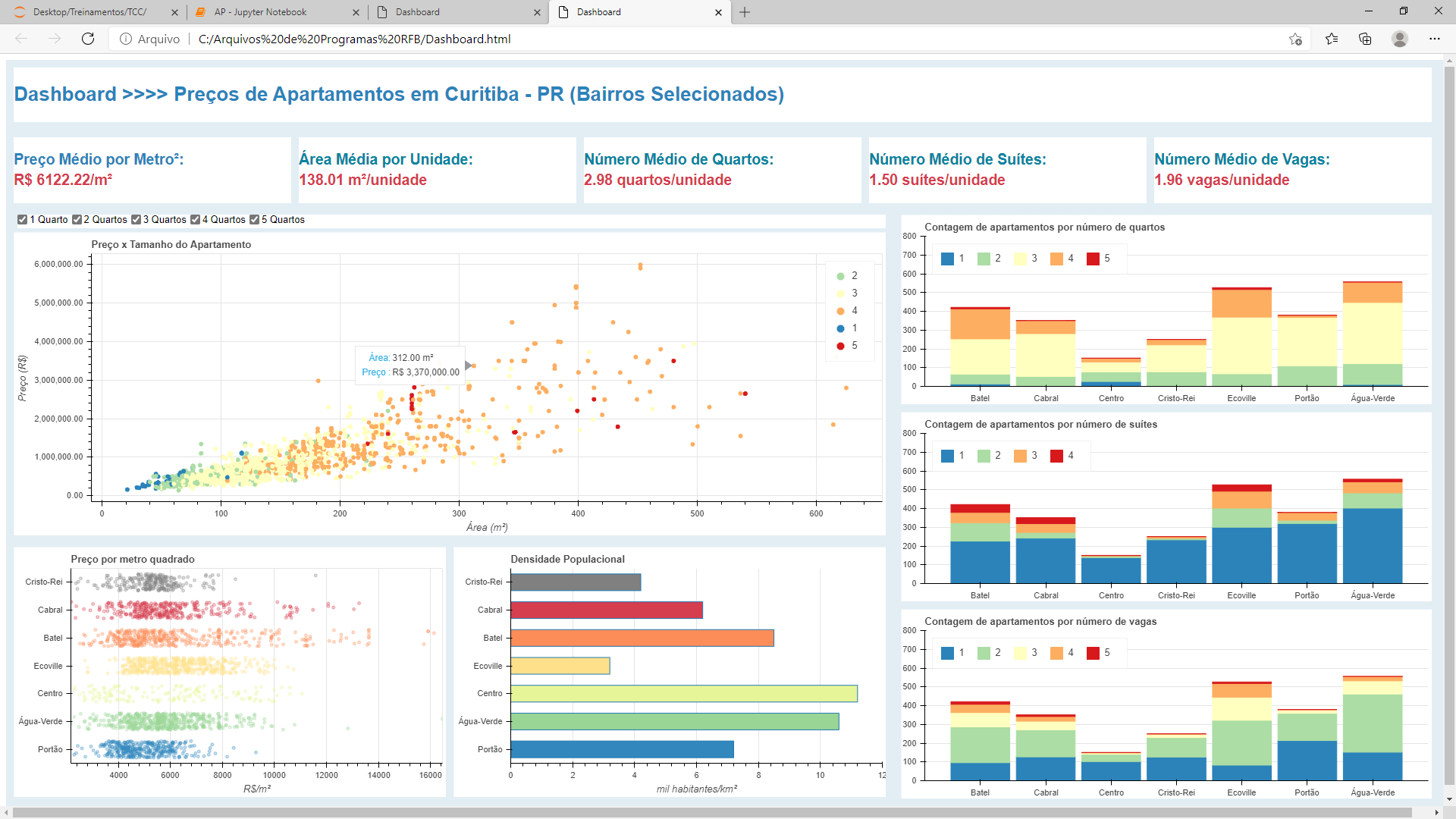
Por fim, temos os códigos para a construção do *layout* do *dashboard* com o encadeamento de colunas e linhas, bem como a exportação do arquivo .html e a abertura dele na página do navegador.

Figura - Código para configuração do layout, geração do arquivo .html e abertura deste no navegador



E na sequência temos o *dashboard* completo e ampliado para melhor visualização, lembrando que esse *dashboard* consta dos anexos deste TCC, no repositório cujo link será informado no próximo tópico, onde é possível interagir com ele e explorar melhor os dados.

Figura - Dashboard completo com hover acionado no gráfico principal



# 7. Links

A seguir estão os links para o repositório deste trabalho e para o vídeo explicativo.

- Link para o vídeo: <https://youtu.be/vySINCDqhaU>

- Link para o repositório dos arquivos: <https://github.com/Alencar2208/TCC-Ciencia-de-Dados-e-Big-Data-2019---PUC-Minas>

# APÊNDICE

**Programação/Scripts**

Anexo I – Jupyter Notebook

1. Disponível em [leg.ufpr.br/~walmes/data/ap\_venda7bairros\_cwb\_210314.txt](http://leg.ufpr.br/~walmes/data/ap_venda7bairros_cwb_210314.txt) [↑](#footnote-ref-1)
2. Disponível em [bairros.pdf (coreconpr.gov.br)](http://www.coreconpr.gov.br/wp-content/uploads/2016/07/bairros.pdf) [↑](#footnote-ref-2)
3. Disponível em [Conjuntos de dados para praticar (ufpr.br)](http://www.leg.ufpr.br/~walmes/cursoR/data-vis/99-datasets.html#apartamentos-a-venda-por-bairro-em-curitiba) [↑](#footnote-ref-3)
4. Disponível em [bairros.pdf (coreconpr.gov.br)](http://www.coreconpr.gov.br/wp-content/uploads/2016/07/bairros.pdf) [↑](#footnote-ref-4)
5. Fonte: [1. Supervised learning — scikit-learn 0.24.1 documentation (scikit-learn.org)](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning) [↑](#footnote-ref-5)
6. Fonte: [Introduction — statsmodels](https://www.statsmodels.org/stable/index.html) [↑](#footnote-ref-6)
7. Disponível em [Conjuntos de dados para praticar (ufpr.br)](http://www.leg.ufpr.br/~walmes/cursoR/data-vis/99-datasets.html#apartamentos-a-venda-por-bairro-em-curitiba) [↑](#footnote-ref-7)
8. Disponível em [pandas - Python Data Analysis Library (pydata.org)](https://pandas.pydata.org/) [↑](#footnote-ref-8)
9. Disponível em [bairros.pdf (coreconpr.gov.br)](http://www.coreconpr.gov.br/wp-content/uploads/2016/07/bairros.pdf) [↑](#footnote-ref-9)
10. Disponível em [tabula-py · PyPI](https://pypi.org/project/tabula-py/) [↑](#footnote-ref-10)
11. A função “corrigir\_nomes” foi obtida no seguinte site [python - Como retirar caractere especial e ponto de coluna string de um data frame? - Stack Overflow em Português](https://pt.stackoverflow.com/questions/217832/como-retirar-caractere-especial-e-ponto-de-coluna-string-de-um-data-frame) [↑](#footnote-ref-11)
12. Documentação disponível em [matplotlib.pyplot — Matplotlib 3.4.1 documentation](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html) [↑](#footnote-ref-12)
13. Documentação disponível em [seaborn: statistical data visualization — seaborn 0.11.1 documentation (pydata.org)](https://seaborn.pydata.org/) [↑](#footnote-ref-13)
14. Fonte: Curso de Ciência de Dados com Python ministrado pelo Prof. Nélio Machado em 10 e 11/2020 (<https://github.com/Alencar2208/DSWP/blob/master/Notebooks/NB15_02__Regress%C3%A3o%20Linear.ipynb>) [↑](#footnote-ref-14)
15. Disponível em [NumPy](https://numpy.org/) [↑](#footnote-ref-15)
16. Dsiponível em [Ordinary least squares - Wikipedia](https://en.wikipedia.org/wiki/Ordinary_least_squares) e [statsmodels.regression.linear\_model.OLS — statsmodels](https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLS.html) [↑](#footnote-ref-16)
17. Disponível em <https://towardsdatascience.com/ridge-lasso-and-elasticnet-regression-b1f9c00ea3a3> [↑](#footnote-ref-17)
18. Disponível em <https://pt.wikipedia.org/wiki/Coeficiente_de_determina%C3%A7%C3%A3o#:~:text=O%20coeficiente%20de%20determina%C3%A7%C3%A3o%2C%20tamb%C3%A9m,observados%20de%20uma%20vari%C3%A1vel%20aleat%C3%B3ria> [↑](#footnote-ref-18)
19. Disponível em <https://seaborn.pydata.org/> [↑](#footnote-ref-19)
20. Disponível em <https://seaborn.pydata.org/examples/many_pairwise_correlations.html> [↑](#footnote-ref-20)
21. Disponível em <https://vas3k.com/blog/machine_learning/?source=post_page-----885aa35db58b----------------------> [↑](#footnote-ref-21)
22. Disponível em <https://en.wikipedia.org/wiki/Akaike_information_criterion> [↑](#footnote-ref-22)
23. Disponível em <https://en.wikipedia.org/wiki/Bayesian_information_criterion> [↑](#footnote-ref-23)
24. Disponível em <https://medium.com/turing-talks/turing-talks-20-regress%C3%A3o-de-ridge-e-lasso-a0fc467b5629> e <https://scikit-learn.org/stable/modules/linear_model.html> [↑](#footnote-ref-24)
25. Disponível em <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html?highlight=lasso#sklearn.linear_model.Lasso> [↑](#footnote-ref-25)
26. Disponível em <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet> [↑](#footnote-ref-26)
27. Disponível em <https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html#sklearn.svm.LinearSVR> [↑](#footnote-ref-27)
28. Disponível em <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor> [↑](#footnote-ref-28)
29. Disponível me <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html?highlight=make_pipeline#sklearn.pipeline.make_pipeline> [↑](#footnote-ref-29)
30. Disponível em <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html?highlight=standardscaler#sklearn.preprocessing.StandardScaler> [↑](#footnote-ref-30)
31. Disponível em <https://docs.bokeh.org/en/latest/index.html> [↑](#footnote-ref-31)
32. Disponível em <https://docs.bokeh.org/en/latest/index.html> [↑](#footnote-ref-32)