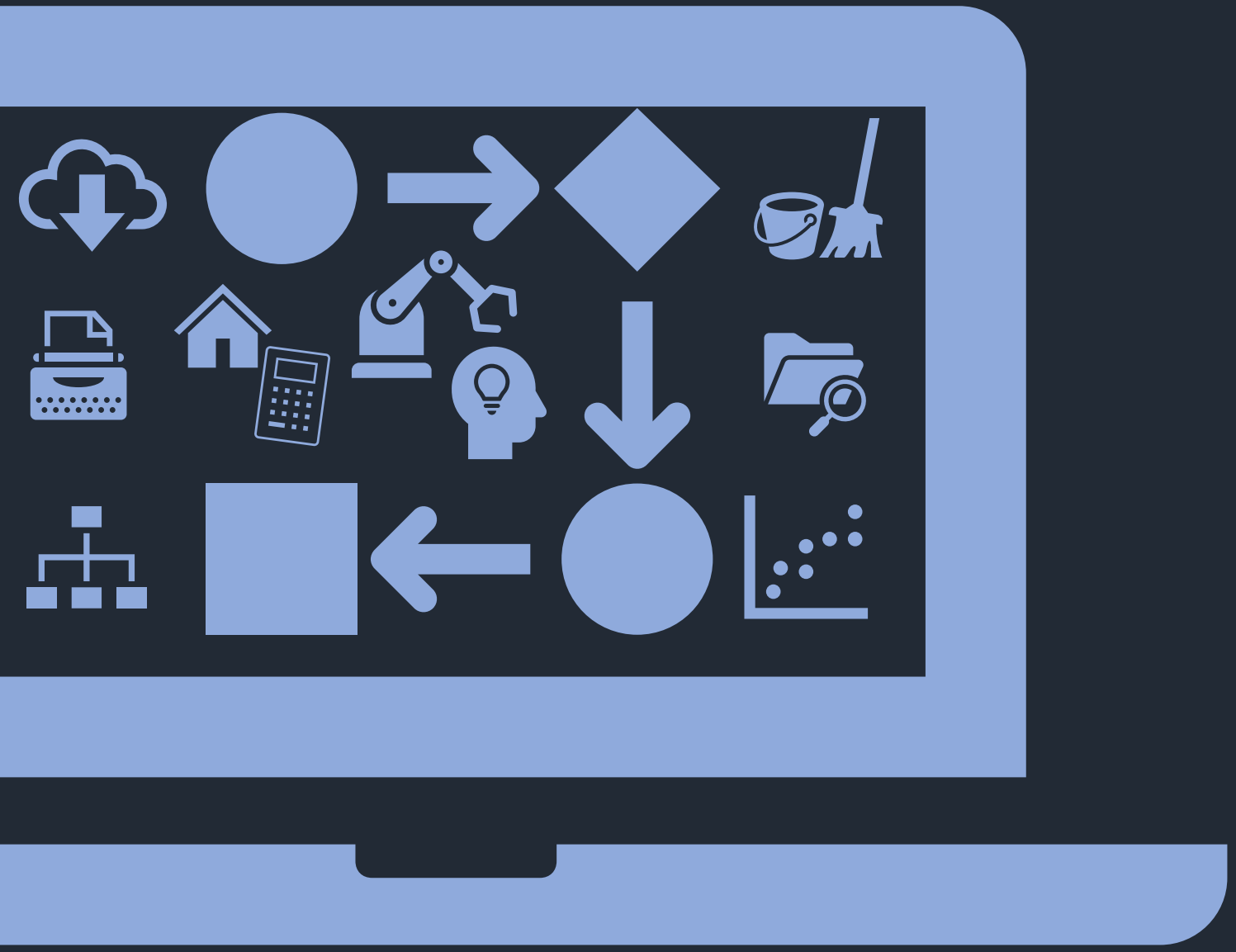


Luiz Alves

Projeto de Data Science

Análise Preditiva Preço/Noite de aluguel por temporada

O que é o projeto?

Tenho um imóvel para locar por temporada, mas qual preço cobrar por diária?

Será que o valor que estou cobrando por diária está acima ou abaixo do praticado pelo mercado?

Quando se tem um imóvel por temporada para alugar sempre há essas dúvidas.

E se houvesse uma forma ou um aplicativo que me dissesse qual o melhor a preço por noite?

Esse é o objeto desse projeto.

Desenvolver um **modelo de predição** que verifique qual o melhor preço possível para locação de imóvel casa/apartamento.

O locatório só terá que informar algumas características e aplicação indicará qual o preço ideal a ser locado.

Metodologia

Coleta dos Dados

Web-Scrapping

airbnb

Anúncios

Limpeza dos Dados

Análise Exploratória

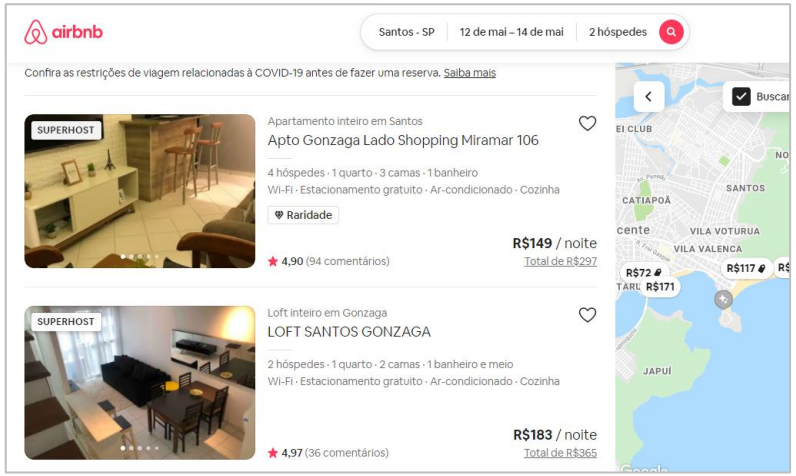
Machine Learning

- *Feature Engineering*
- *Data Preparation*
- Modelagem

Deploy

Serviço de calculadora de aluguel disponível na web.

Coleta dos dados: *web-scrapping*



Seleção dos pontos de interesse no anúncio

Munícipios: Santos, São Vicente, Guarujá, Bertioga, Praia Grande, Mongaguá, Itanhaém e Peruíbe).

É desenvolvida uma função que realiza à extração em massa.

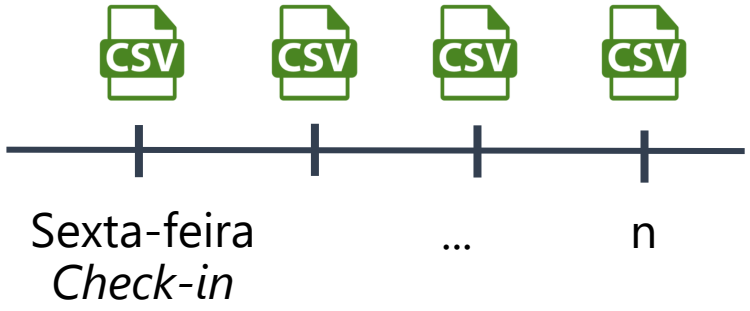
id	check-in	check-out	titulo	titulo_2	atributo_1	atributo_2	atributo_3	atributo_4	atributo_5	atributo_6	atributo_7	atributo_8	avaliação
41129581	2021-03-12	2021-03-14	Quarto inteiro em Peruíbe	Pousada Suíte c/ Piscina e Ar Cond - 100m do mar	2 hóspedes	1 quarto	1 cama	1 banheiros privado	Wi-Fi	Ar-condicionado	Piscina	Estacionamento gratuito	4.91

Coleta dos dados: *web-scrapping*

Como é o processo...

Web-Scrapping

airbnb



3 meses de coletas

Foi programada a coleta com Check-in as sextas-feiras e Check-out aos domingo por um período de três meses.

Ao final do processo, todos os arquivos foram concatenados formando a base de dados.

CSV

7675 linhas

19 colunas

Código

Web-scrapping

```
# 2 opções de URL
urlperuibe = 'https://www.airbnb.com.br/s/Peru%C3%ADbe-~-SP/homes?tab_id=home_tab&r

url=urlperuibe

# Para enganar o site, permutaremos entre 2 opções de assinaturas de browser.

userAgents=[
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Headles
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/605.1.15 (KHTML, l

doc = requests.get(url.format(npagina=1), headers={"User-agent": userAgents[1]})

analizador = BeautifulSoup(doc.content, 'html.parser')

imoveis = analizador.find_all('div', class_="_gig1e7")
```


```
# em rows guardo uma lista temporária de dicts de dados capturados, 1 dict por imóvel
imoveis_=[]

for pagina in imoveis:
    #print("Estou aqui: " + url.format(npagina=pagina))

    # pega a página do site pela internet
    for i in pages:
        doc = requests.get(i, headers={"User-agent": userAgents[1]})

        # analiza o HTML
        analizador = BeautifulSoup(doc.content, 'html.parser')

        # extrai somente a lista de imóveis (em HTML) usando o seletor descoberto no código da págin
        imoveis = analizador.find_all('div', class_="_gig1e7")

        for unidade in imoveis:
            uni={}
            # extrai dado por dado segundo seus seletores...

            # id:
            for link in unidade.select("a"):
                rooms = link.get('href')
                rooms = rooms.split('?')
                uni['id'] = rooms[0].replace('/rooms/', ' ').strip()
                #quarto:
                rooms = rooms[1].split('&')
                #check-in:
                uni['check-in'] = rooms[1].replace('check_in=', ' ').strip()
                #check-out:
                uni['check-out'] = rooms[2].replace('check_out=', ' ').strip()


            # titulo 1:
            uni['titulo'] = unidade.find("div",class_="_tmwq9g").find("div",class_="_1jzvdu8")
                            .find("div",class_="_1tanv1h").find("div",class_="_b14dlit")
                            .contents[0].strip()

            # titulo 2:
            uni['titulo_2'] = unidade.find("div",class_="_tmwq9g").find("div",class_="_bzh5lkq")
                              .contents[0].strip()

            # atributo 1:
            try:
                uni['atributo_1']  = unidade.find("div",class_="_kqh46o").contents[0].strip()
            except (IndexError, ValueError, AttributeError):
                continue

            # atributo 2:
            try:
                uni['atributo_2'] = unidade.find("div",class_="_kqh46o").contents[2].strip()
            except (IndexError, ValueError, AttributeError):
                continue
```

Amostra da função de extração em massa

Limpeza de Dados

O processo de limpeza foi divido em **03 etapas.**

Primeira Limpeza de dados

- Renomear colunas
- Eliminar linhas duplicadas – De 7675 para 5262 (Removidas 2413)
- Verificar inconsistências (datas erradas)
- Colunas constantes

Apenas números nas variáveis numéricas. *Replace*

atributo_1	atributo_2	atributo_3	atributo_4
3 hóspedes	2 quartos	2 camas	1 banheiro
4 hóspedes	1 quarto	2 camas	1 banheiro
3 hóspedes	2 quartos	2 camas	1 banheiro

Número Hóspedes	Número Quartos	Número Camas	Número Banheiros
3	2	2	1
4	1	2	1
3	2	2	1

Apenas o nome do bairro

titulo
Condomínio inteiro em Gonzaga
Apartamento inteiro em Gonzaga
Condomínio inteiro em Gonzaga

Localização
Gonzaga
Gonzaga
Gonzaga

Limpeza de Dados

Segunda Limpeza de dados

Por algum motivo os valores de preço vieram com erros a partir do Scrapping.

O era para ser 1022.0 está como 1.022

Dado este erro foi necessário criar uma função que filtrasse os valores que estão errados aplicasse a correção nessas linhas.

As variáveis que tiveram esse erro foram Preço/noite e Preço com taxa.

Função de correção

```
def corrigir_monetario(df, name='name'):
    """
    Correção de valores float incorretos 1.0 para 1000.0
    df = Dataframe
    name= Nome da variável
    """

    filter_1 = df.loc[(df[name] < 10.0), :].index
    df['New_name'] = df.iloc[filter_1][name] * 1000
    value = df[name]
    df['New_name'] = df['New_name'].fillna(value)
    df = df.drop(name, axis=1)
    df.rename(columns = {'New_name': name}, inplace=True)

    return df
```

```
df_test_funccion = corrigir_monetario(df_test_funccion, name='Preço/Noite')
```

```
df_test_funccion = corrigir_monetario(df_test_funccion, name='Preço com taxas')
```

Limpeza de Dados

Terceira Limpeza de dados

- Linhas e colunas duplicadas
- Colunas constantes
- Ajustes dos tipos (*int, float, objeto, datetime*)

Dummies para variáveis categóricas

Plus 1	Plus 2	Plus 3	Plus 4
Ar-condicionado	Piscina	Cozinha	Estacionamento gratuito
Wi-Fi	Cozinha	Permitido animais	Elevador
Wi-Fi	Ar-condicionado	Cozinha	Estacionamento gratuito

Vista para o mar	Piscina	Lava-louças	Wi-Fi	Café da Manhã	Máquina de Lavar	Estacionamento gratuito	Academia	Jacuzzi	Secadora	Entrada/saída para esquis	Cozinha
0	1	0	1	0	0	0	0	0	0	0	1
0	1	0	1	0	0	0	0	0	0	0	1
0	0	0	1	0	1	1	0	0	0	0	1

Código

Criar *Dummies*

```
def create_dummies(df):
    # Dummies das colunas Plus
    plus1 = df['Plus 1'].unique()
    plus2 = df['Plus 2'].unique()
    plus3 = df['Plus 3'].unique()
    plus4 = df['Plus 3'].unique()
    fullplus = plus1.tolist() + plus2.tolist() + plus3.tolist() + plus4.tolist()
    fullplus = set(fullplus)
    def word_in_columns(df, word):
        if word in df['Plus 1'] or word in df['Plus 2'] or word in df['Plus 3'] or word in df['Plus 4']:
            return 1
        else:
            return 0
    count = 0
    for word in fullplus:
        df[f'{word}'] = df.apply(word_in_columns, axis=1, args= (word, ))
        count += 1
    # Dummies para Superhost
    Superhost = df['Superhost'].unique()
    def host_in_columns(df, word):
        if word in df['Superhost']:
            return 1
        else:
            return 0
    count = 0
    for word in Superhost:
        df[f'{word}'] = df.apply(word_in_columns, axis=1, args= (word, ))
        count += 1
    #Drop das colunas
    df = df.drop(columns=['Plus 1' , 'Plus 2', 'Plus 3', 'Plus 4', 'Superhost'])

    return df
```

```
df_train = create_dummies(df_train)
```

Limpeza de Dados

Terceira Limpeza de dados

```
df_train['Número Banheiros'].value_counts()
```

```
1                          2674
2                           764
3                           191
1 e meio                    189
1 privado                   171
2 e meio                     93
4                            29
3 e meio                     28
1 compartilhado              18
5 e meio                     16
5                             9
2 compartilhados              7
4 e meio                      6
0                             4
6 e meio                      4
6                             4
1 compartilhado e meio        2
Name: Número Banheiros, dtype: int64
```

Em alguns imóveis observou que existem banheiros compartilhados.

Se aplicarmos uma função que retirasse o texto, perderíamos essa informação. A solução neste caso foi criar uma nova coluna informando se há banheiro compartilhado no imóvel através de uma função.

```
def banheiro_transformation(df):

    # Pega o qualquer texto após o espaço.
    df['Banheiro Compartilhado'] = df['Número Banheiros'].str.contains('\\ ', regex=True)

    # Pega o primeiro digito numérico
    df['Número Banheiros'] = df['Número Banheiros'].str[0:1]
    df['Número Banheiros'] = df['Número Banheiros'].str.strip()

    return df
```

```
df_train = banheiro_transformation(df_train)
```

Limpeza de Dados

Terceira Limpeza de dados

Valores Atípicos: Há quartos em que não 0 camas e 0 Banheiros

Investigando o número de camas.

```
# Invetigando o número de camas.
df_train['Número Camas'].value_counts()
```

```
2     1195
3      833
1      825
4      581
5      214
6      192
7      126
8       68
9       65
10      34
0       22
11      19
16      15
12      12
15       8
Name: Número Camas, dtype: int64
```

Podemos supor que se há quartos, há camas?

```
# Supondo que se há quarto é possivel que haja cama.
len(df_train.loc[(df_train['Número Camas'] == 0 & (df_train['Número Quartos'] >=1))])
```

```
22
```

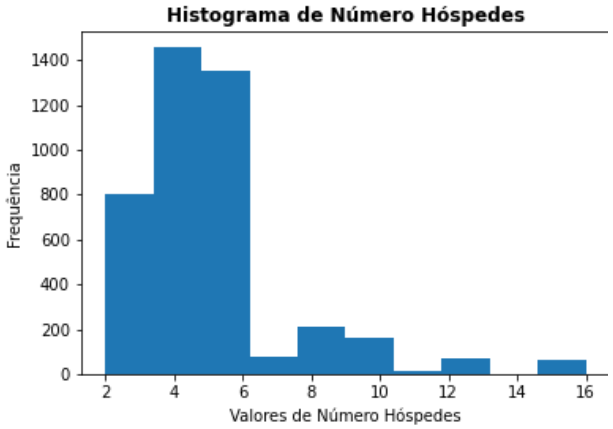
Entretanto, Existe a possibilidade do locatário alugar o imóvel por temporada sem disponibilizar uma cama. O cliente pode levar uma cama inflável por exemplo.

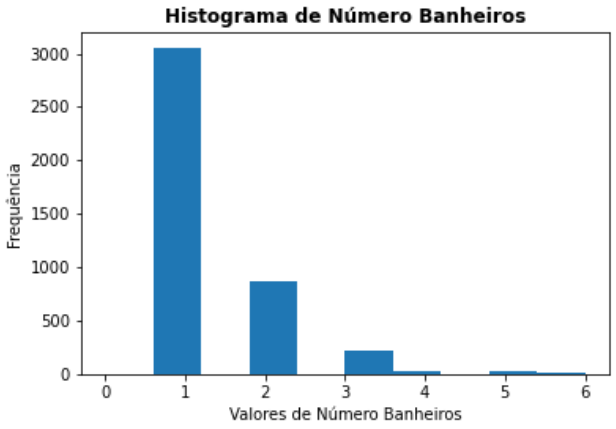
Pesando nesta linha, vamos optar por não alterar e/ou remover estes imóveis.

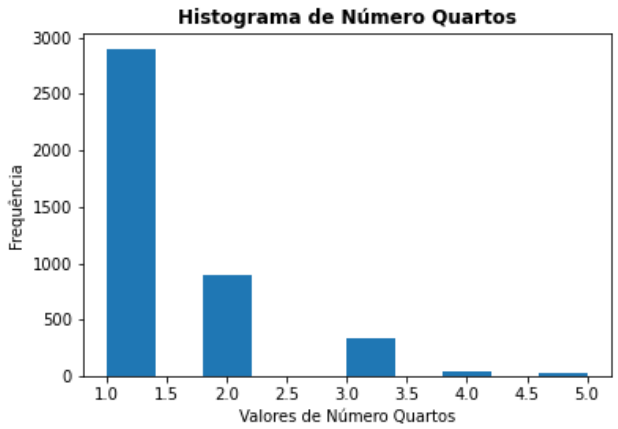
No caso de haver 0 banheiros podemos considerar como sendo banheiro compartilhado.

Valores Atípicos: Número de quartos com a nomenclatura estúdio foi considerado como 01 dormitório..

Análise Exploratória







Frequências nos dados

Santos/SP

1 quarto

1 Banheiro

25 comentários

Capacidade
4 a 6

2 camas

Avaliação
+ 4,75 (1 – 5)

R$ 100 e R$ 200
(1 diária sem taxa)

Código

```
col_num = ['Número Hóspedes', 'Número Quartos', 'Número Camas', 'Número Banheiros',
           'Avaliação', 'Número Comentários', 'Preço com taxas']
col_cat = ['Academia', 'Ar-condicionado', 'Cozinha', 'Elevador', 'Estacionamento gratuito',
           'Máquina de Lavar', 'Permitido animais', 'Piscina', 'Secadora', 'Self check-In',
           'Vista para as águas', 'Vista para o mar', 'Wi-Fi', 'Café da Manhã', 'Entrada/saída para esquis',
           'Jacuzzi', 'Lareira interna','Lava-louças', 'Novo preço mais baixo', 'Raridade', 'Localização']
col_date = ['Check-In', 'Check-Out']
Target = ['Preço/Noite']
```

Histogramas

```
col_cat.remove('Localização')

for colunas in col_num + col_cat + Target:

    plt.figure(figsize=(6, 4))
    plt.hist(df_train[colunas])

    plt.title(f'Histograma de {colunas}', fontsize=12, fontweight='bold')
    plt.xlabel(f'Valores de {colunas}')
    plt.ylabel(f'Frequência')

    #plt.savefig(f'../img/hist_{colunas}.png')

    plt.show()
```

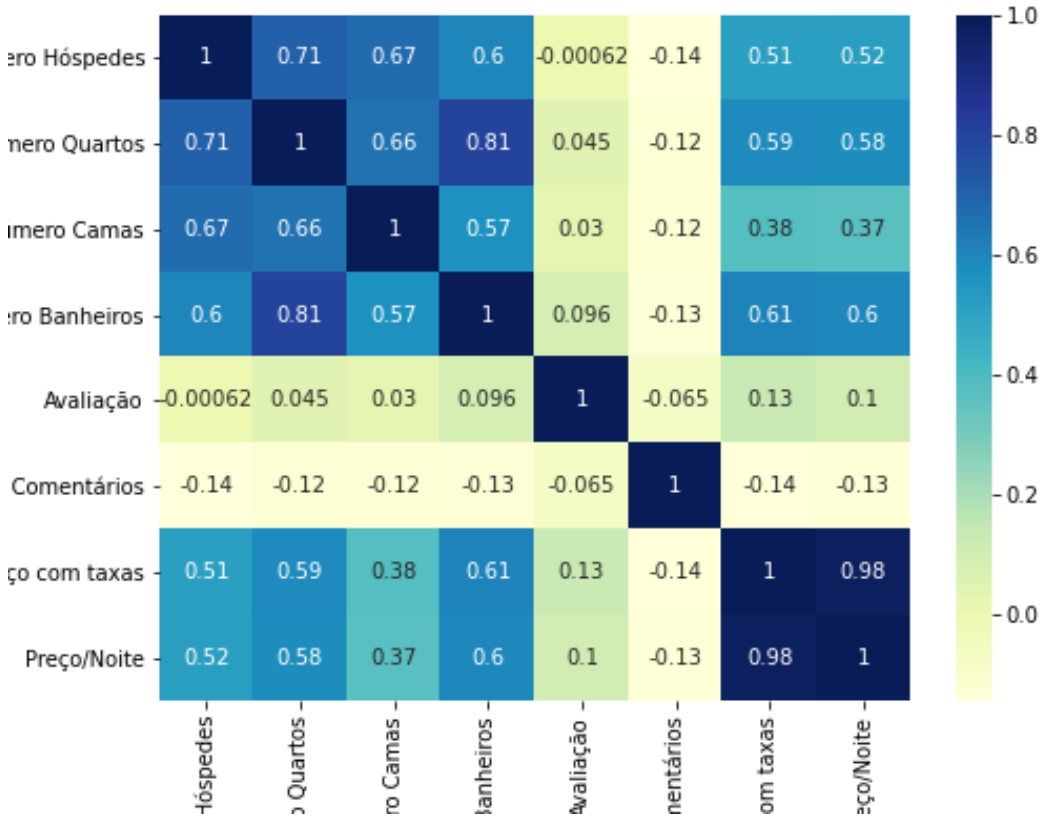
Análise Exploratória

Matriz de Correlação

Na matriz de correlação podemos verificar o quão correlacionadas estão as variáveis numéricas com relação ao target (Preço/Noite).

É possível afirmar estatisticamente que há correção positiva entre:

Número Hóspedes, Quartos, Camas, Banheiros, Comentários e Avaliações.



Código

```
col_num = ['Número Hóspedes', 'Número Quartos', 'Número Camas', 'Número Banheiros',
           'Avaliação', 'Número Comentários', 'Preço com taxas']
col_cat = ['Academia', 'Ar-condicionado', 'Cozinha', 'Elevador', 'Estacionamento gratuito',
           'Máquina de Lavar', 'Permitido animais', 'Piscina', 'Secadora', 'Self check-In',
           'Vista para as águas', 'Vista para o mar', 'Wi-Fi', 'Café da Manhã', 'Entrada/saída para esquis',
           'Jacuzzi', 'Lareira interna','Lava-louças', 'Novo preço mais baixo', 'Raridade', 'Localização']
col_date = ['Check-In', 'Check-Out']
Target = ['Preço/Noite']
```

Gráfico da Matriz de Correlação

```
# Matriz de Correlação

correlacao = df_train[col_num + Target].corr()
plt.figure(figsize=(8,6))
sns.heatmap(correlacao, cmap="YlGnBu", annot=True)
plt.savefig(f'../img/matriz_correlacao.png')
plt.show()
```

Teste de Correlação - Spearman

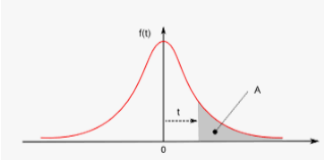
```
# Estatísticamente
colunas = ['Número Hóspedes', 'Número Quartos', 'Número Camas',
           'Número Banheiros', 'Avaliação', 'Número Comentários']
correlation_features = []
for col in colunas:
    cor, p = stats.spearmanr(df_train[col],df_train[Target])
    if p <= 0.5:
        correlation_features.append(col)
        print(f'p-value: {p},  correlation: {cor}')
        print(f'Existe correlação entre {col} e {Target}.')
        print('--'*30)
    else:
        print(f'p-value: {p},  correlation: {cor}')
        print(f'Não há correlação entre {col} e {Target}.')
        print('--'*30)
```

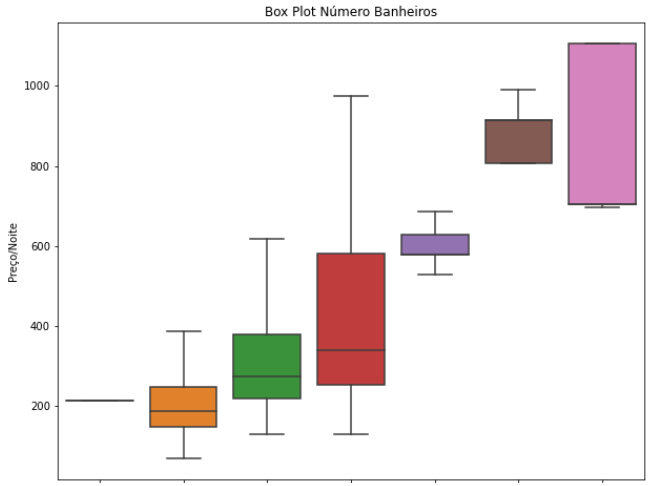
Análise Exploratória

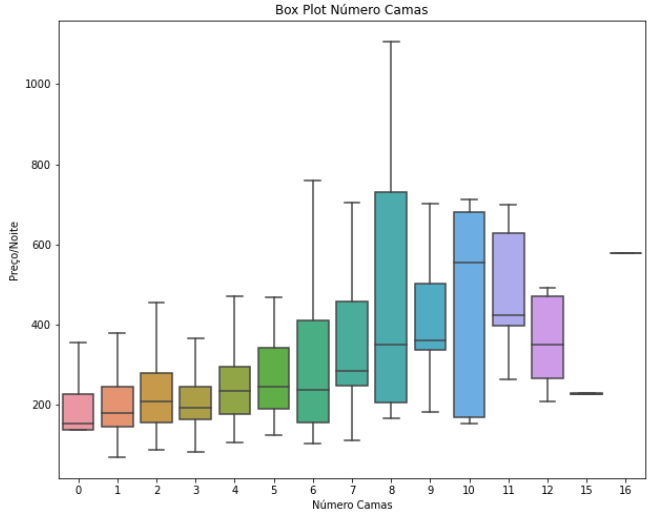
Teste de Hipóteses

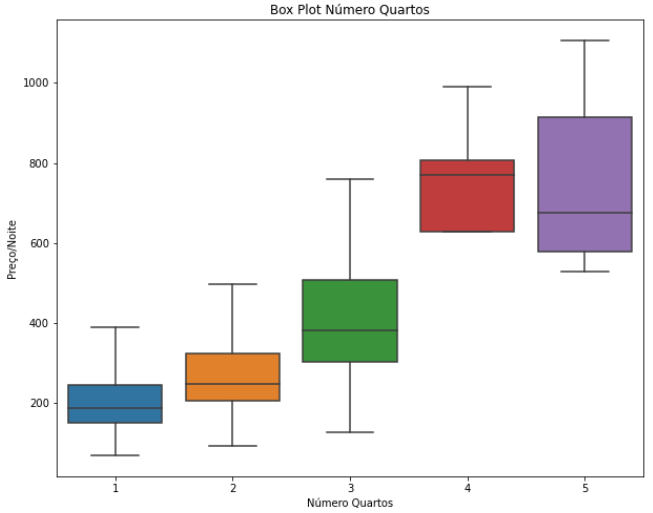
H1. Imóveis com maior número de atributos (número de Hospedes, Quartos, Camas e banheiro) possuem maior preço/noite maior?

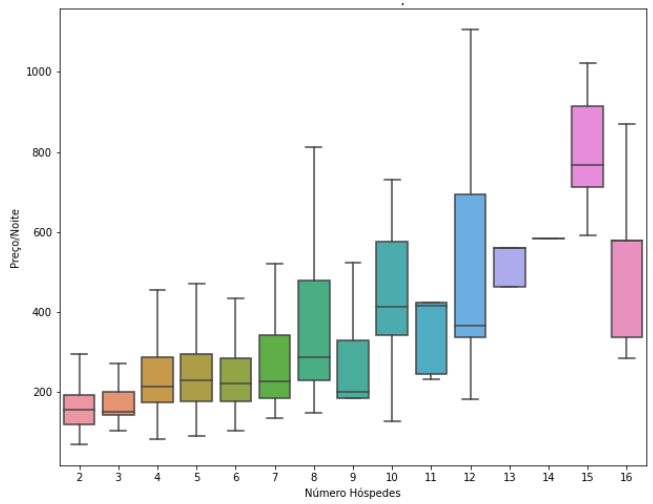
Graficamente e pelo teste t-Student podemos afirmar que quanto maior o tamanho do imóvel maior o preço/noite











Código

```
atributos = ['Número Hóspedes', 'Número Quartos', 'Número Camas', 'Número Banheiros']
Target = ['Preço/Noite']
```

Gráfico BoxPlot:

```
for colunas in atributos:
    plt.figure(figsize=(10,8))
    sns.boxplot(x=df_train[colunas],
                y= 'Preço/Noite',
                data=df_train,
                showfliers=False)

    plt.title(f'Box Plot {colunas}')
    plt.xlabel(colunas)
    plt.ylabel('Preço/Noite')

    #plt.savefig(f'../img/box_plot_atributos_{colunas}.png')

    plt.show()
```

Teste t-Student

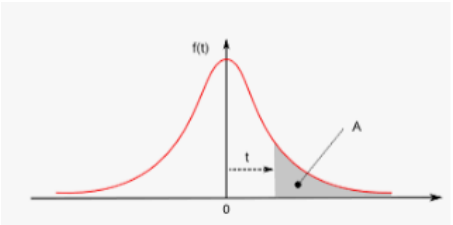
```
def teste_t(df, features, target='0', alpha=0.05):
    """
    Teste T-Student
    df = DataFrame
    fearures = List of columns to be tested
    target = 'Target'
    alpha = significance index. Default is 0.05
    """
    import scipy
    for colunas in features:
        true_target = df.loc[df[colunas] == 1, [target]]
        false_target = df.loc[df[colunas]==0, [target]]
        teste_T_result = scipy.stats.ttest_ind(true_target, false_target, equal_var=False)
        if teste_T_result[1] < alpha:
            print(f'{colunas}: ')
            print(f'{teste_T_result}')
            print(f'Não')
            print('-'*30)
        else:
            print(f'{colunas}: ')
            print(f'{teste_T_result}')
            print(f'Sim - O preço é maior')
            print('-'*30)
```

```
teste_t(df_train, atributos, target='Preço/Noite')
```

Análise Exploratória

Teste de Hipóteses

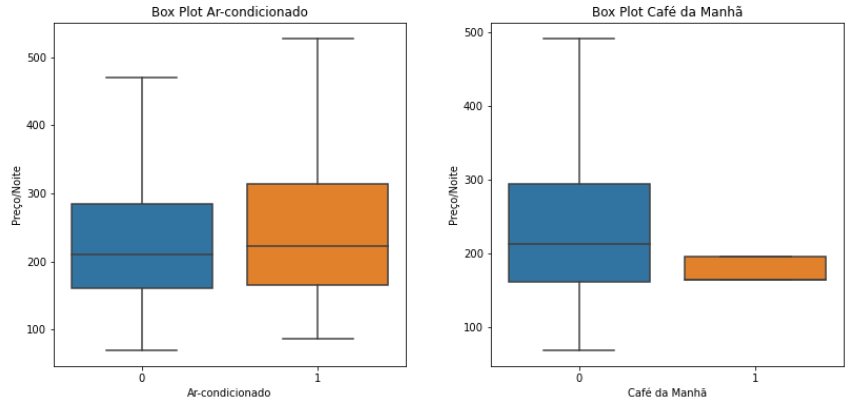
H2. Imóveis com comodidades tem o preço/noite maior?



Teste Gráfico e Estatístico t-Student

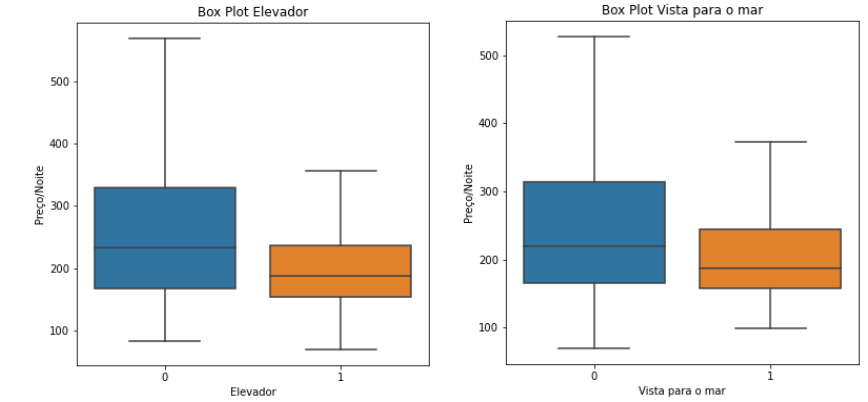
Impactam no preço:

Estacionamento gratuito, máquina de lavar, secadora, vista para águas, entrada/saída para esquis, novo preço mais baixo e Raridade



Não impactam no preço:

Academia, Ar-condicionado, cozinha, elevador, permitido animais, piscina, self- Check-In, Vista para o mar, wi-fi, café da manhã, Jacuzzi, lareira interna e lava-louças.



Código

```
col_cat = ['Academia', 'Ar-condicionado', 'Cozinha', 'Elevador', 'Estacionamento gratuito',
           'Máquina de Lavar', 'Permitido animais', 'Piscina', 'Secadora', 'Self check-In',
           'Vista para as águas', 'Vista para o mar', 'Wi-Fi', 'Café da Manhã', 'Entrada/saída para esquis',
Target = ['Preço/Noite']
```

Gráfico BoxPlot:

```
for colunas in col_cat:
    plt.figure(figsize=(6,6))
    sns.boxplot(x=df_train[colunas],
                y='Preço/Noite',
                data=df_train,
                showfliers=False)

    plt.title(f'Box Plot {colunas}')
    plt.xlabel(colunas)
    plt.ylabel('Preço/Noite')

    plt.savefig(f'../img/box_plot_comodidades_{colunas}.png')
    plt.show()
```

Teste t-Student

```
def teste_t(df, features, target='0', alpha=0.05):
    """
    Teste T-Student
    df = DataFrame
    fearures = List of columns to be tested
    target = 'Target'
    alpha = significance index. Default is 0.05
    """
    import scipy
    for colunas in features:
        true_target = df.loc[df[colunas] == 1, [target]]
        false_target = df.loc[df[colunas]==0, [target]]
        teste_T_result = scipy.stats.ttest_ind(true_target, false_target, equal_var=False)
        if teste_T_result[1] < alpha:
            print(f'{colunas}: ')
            print(f'{teste_T_result}')
            print(f'Não')
            print('-'*30)
        else:
            print(f'{colunas}: ')
            print(f'{teste_T_result}')
            print(f'Sim - O preço é maior')
            print('-'*30)
```

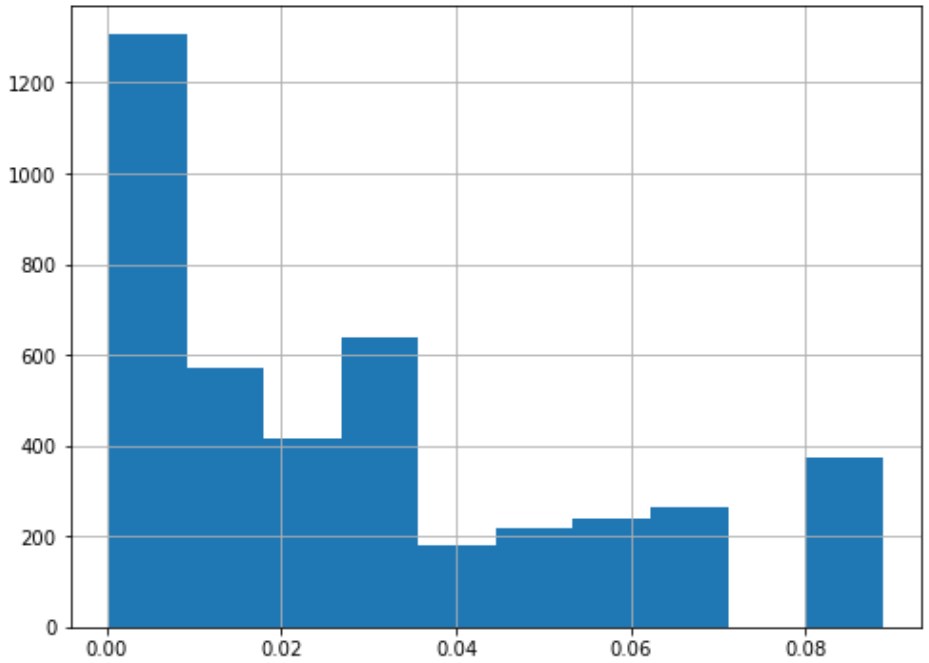
```
teste_t(df_train, atributos, target='Preço/Noite')
```

Análise Exploratória

Teste de Hipóteses

H3. Existe correlação entre a localização e o Preço/Noite?

É possível observar o preço/noite tende a ser menor em locais com maior concentração de imóveis disponíveis.
Ou seja, quanto maior a oferta menor o preço.



Distribuição de frequência de localização

Código

Gráfico Histograma:

```
# Usando o transformador de Frequency Encounder.
cfce = CountFrequencyEncoder(encoding_method='frequency', variables=['Localização'])
df_train_transf = cfce.fit_transform(df_train)

#Gráfico:
plt.figure(figsize=(8,6))
df_train_transf['Localização'].hist()
plt.show()
```

Correlação:

```
# H3: Existe Correlação entre a localização e o Preço/Noite?
local = ['Localização', 'Preço/Noite']
correlacao_local = df_train_transf[local].corr()['Preço/Noite']
correlacao_local
```

Teste de Correlação:

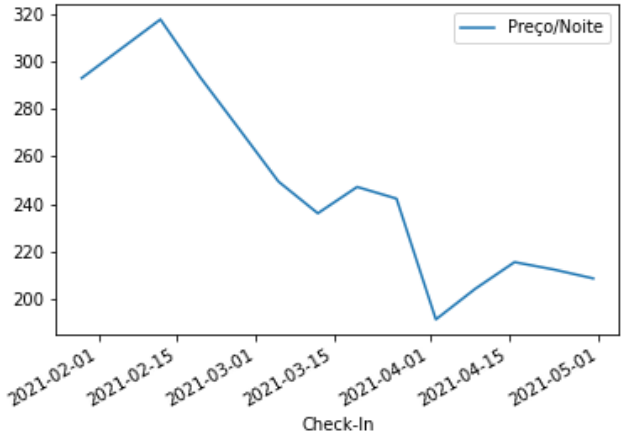
```
# vamos testar estatísticamente:
cor, p = stats.spearmanr(df_train_transf['Localização'],df_train_transf['Preço/Noite'])
if p <= 0.5:
    print(f'p-value: {p},  correlation: {cor}')
    print(f'Existe correlação.')
else:
    print(f'p-value: {p},  correlation: {cor}')
    print(f'Não há correlação.')
```

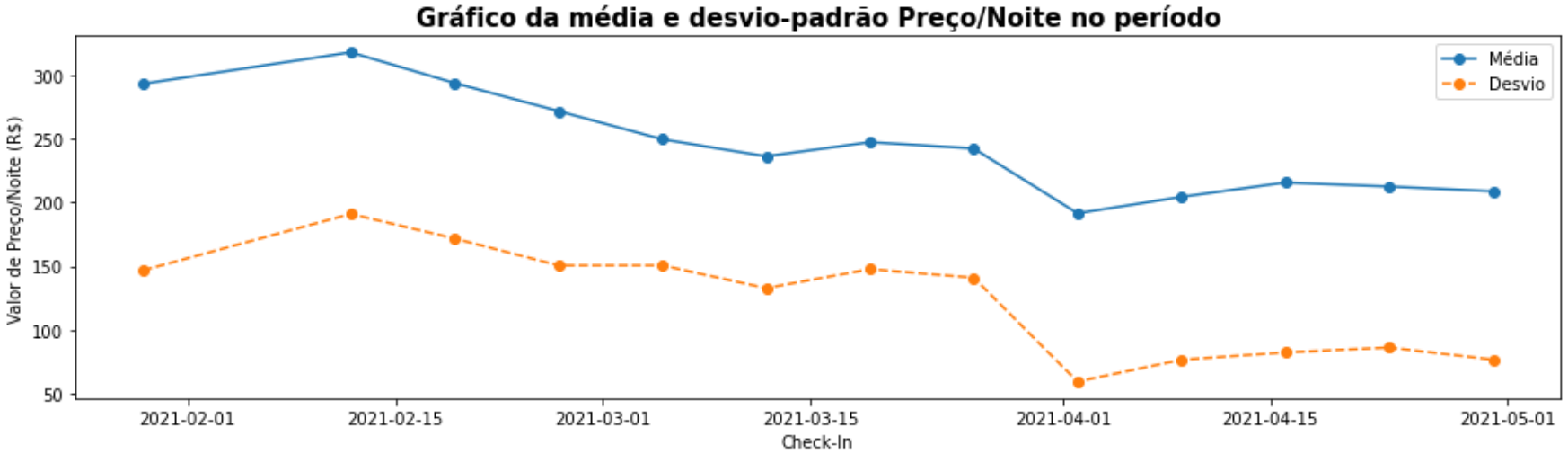
```
p-value: 4.3741112668780545e-08,  correlation: -0.08426624393023412
Existe correlação.
```

Análise Exploratória

Teste de Hipóteses

H4. Qual o comportamento dos preços/noite ao longo da coleta?





Carnaval

Páscoa

É possível notar uma queda acentuada no dia 02/04 - motivo - Feriado Sexta-feira Santa. Porém, não é possível afirmar este comportamento durante o ano todo.

Todavia, é possível afirmar que, sendo a data de check-In coincidindo no mesmo dia do feriado temos uma redução de preço médio.

Código

```
df_preco_medio = df_train.groupby('Check-In').agg(Mean = ('Preço/Noite', 'mean'),
                                                  Desvio = ('Preço/Noite', 'std'),
                                                  Median = ('Preço/Noite', 'median'),
                                                  Min = ('Preço/Noite', 'min'),
                                                  Max = ('Preço/Noite', 'max')).reset_index()
```

Gráfico linhas 1:

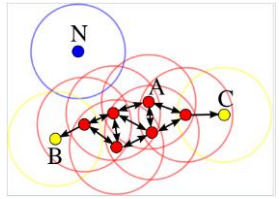
```
df_train.pivot_table(values='Preço/Noite',
                     columns='Check-In',
                     aggfunc='mean').T.plot()
plt.show()
```

Gráfico linhas 2:

```
plt.figure(figsize=(16, 4))

#faz o plot da linha
plt.plot(df_preco_medio['Check-In'],
        df_preco_medio['Mean'],
        'o-',
        label='Média')

plt.plot(df_preco_medio['Check-In'],
        df_preco_medio['Desvio'],
        'o--',
        label='Desvio')

# Adiciona títulos
plt.title(
    'Gráfico da média e desvio-padrão Preço/Noite no período',
    fontsize=15,
    fontweight='bold')

plt.xlabel('Check-In')
plt.ylabel('Valor de Preço/Noite (R$)')

plt.legend()

plt.savefig(f'../img/media_preço_noite_por_periodo.png')

plt.show()
```

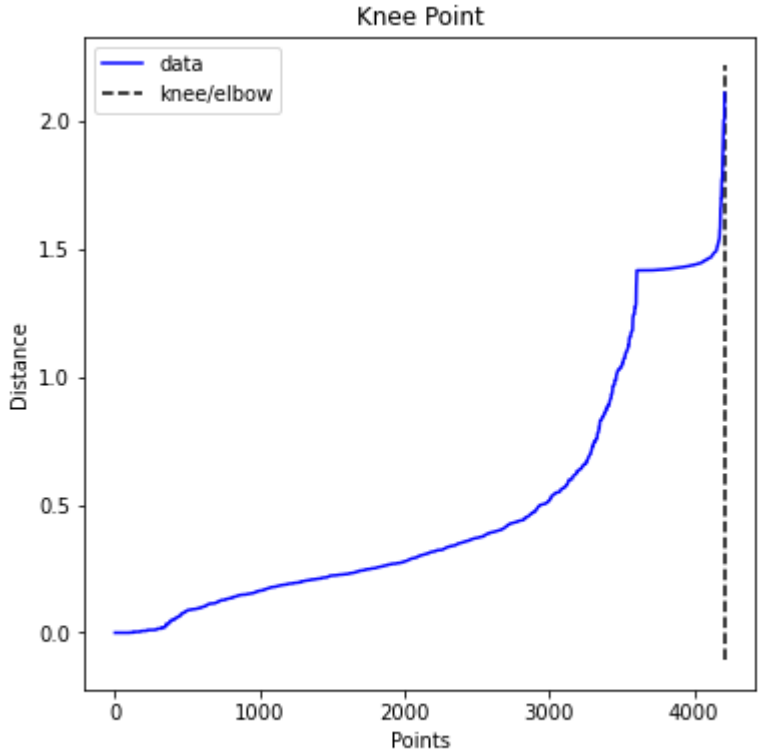
Análise Exploratória

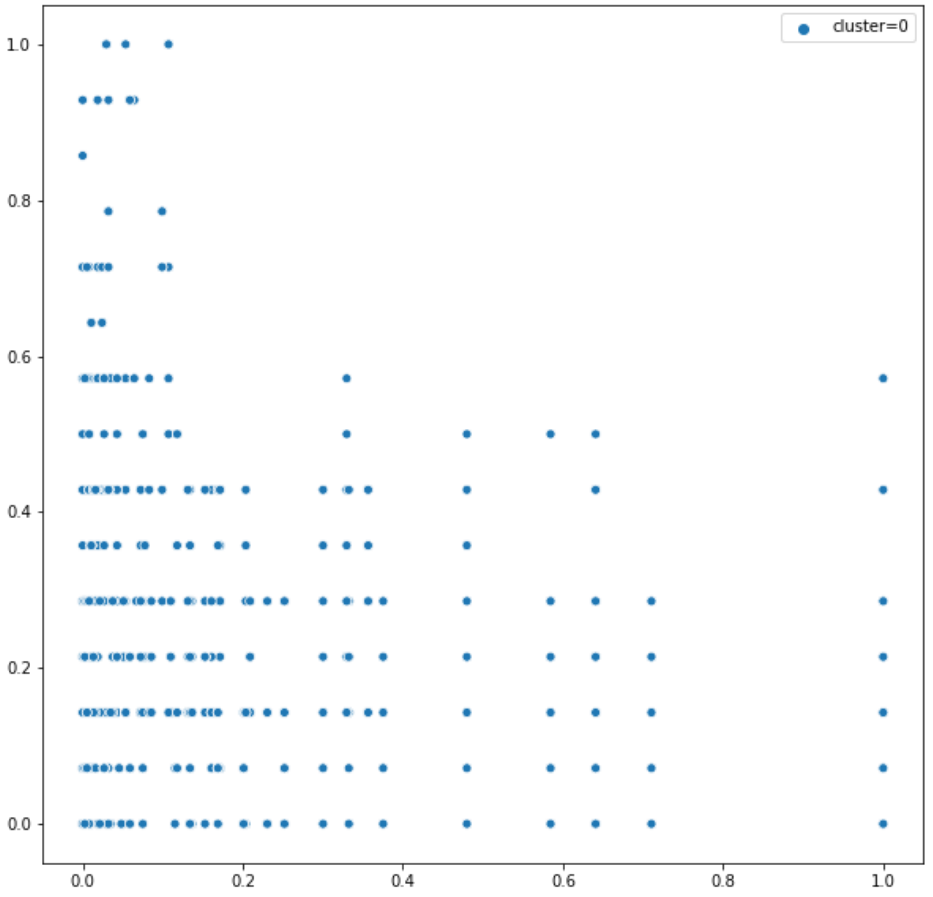
Teste de Hipóteses



H5. Há clusters ou outliers?

Pelo método de clusterização DBSCAN, Podemos perceber que não há grupos nesse conjunto de dados.





2.1066622572159717

Código

Knee Point e Gráfico de Cluster DBSCAN:

```
X = df_train.drop({'ID', 'Check-In', 'Check-Out'}, axis=1)
cfce = CountFrequencyEncoder(encoding_method='frequency', variables=['Localização'])
pipe = Pipeline(steps=[('scaler', MinMaxScaler())])

X = cfce.fit_transform(X)
X = pipe.fit_transform(X)

# DBSCAN
nearest_neighbors = NearestNeighbors(n_neighbors=11)
neighbors = nearest_neighbors.fit(X)

distances, indices = neighbors.kneighbors(X)
distances = np.sort(distances[:,10], axis=0)

i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp_method='polynomial')
fig = plt.figure(figsize=(5, 5))
knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")
plt.show()
print(distances[knee.knee])

#Gráfico
db = DBSCAN(eps=distances[knee.knee], min_samples=11).fit(X)
labels = db.labels_
fig = plt.figure(figsize=(10, 10))
sns.scatterplot(X[:,0], X[:,1], hue=["cluster={}".format(x) for x in labels])
plt.show()
```

Feature Engineering

A partir da Análise Exploratória de Dados (EDA), serão criadas novas features para explicar este modelo.

- Agrupar vista para águas e vista para o mar,
- Transformar o Check-in em dia, mês e ano.
- Há Alta demanda de locais disponíveis naquele local?
- É feriado no dia do check-In?
- Há feriado na semana do check-In?
- É outlier?

Código

Criar coluna Demanda:

```python
def eng_create_demand(df):
    """
    Create new a column called Demanda from maximum localization
    df= dataset
    """
    df['Demanda'] = df['F Localização']
    df['Demanda'] = [1 if i == df['F Localização'].max()
                     else 0 for i in df['Demanda']]
    return df
```

```python
df_train = eng_create_demand(df_train)
```

Criar coluna é feriado

```python
df_feriados = pd.read_csv('../data/feriados_nacionais_2021.csv', sep=';')
```

```python
def eng_create_is_holiday(df , df_feriados):
    """
    Create new column called É feriado.
    df = Dataframe
    df_feriados = Dafaframe contendo uma lista de feriados nacionais

    """
    #import da tabela feriado
    df_feriados = df_feriados.drop('evento', axis=1)
    df_feriados.replace({'feriado nacional' : '1', 'ponto facultativo': '1'}, inplace=True)
    df_feriados.rename(columns={'status': 'É_feriado'}, inplace=True)
    df_feriados.rename(columns={'data': 'Check-In' }, inplace=True)
    df_feriados['Check-In'] = pd.to_datetime(df_feriados['Check-In'], format ='%Y-%m-%d')

    # Vamos juntar as duas tabelas Preço Médio e Feriados
    df = df.merge(df_feriados, left_on='Check-In', right_on='Check-In', how='left')

    #preenche os nulos com 0
    df = df.fillna(0)

    return df
```

```python
df_train = eng_create_is_holiday(df_train, df_feriados)
```

Código

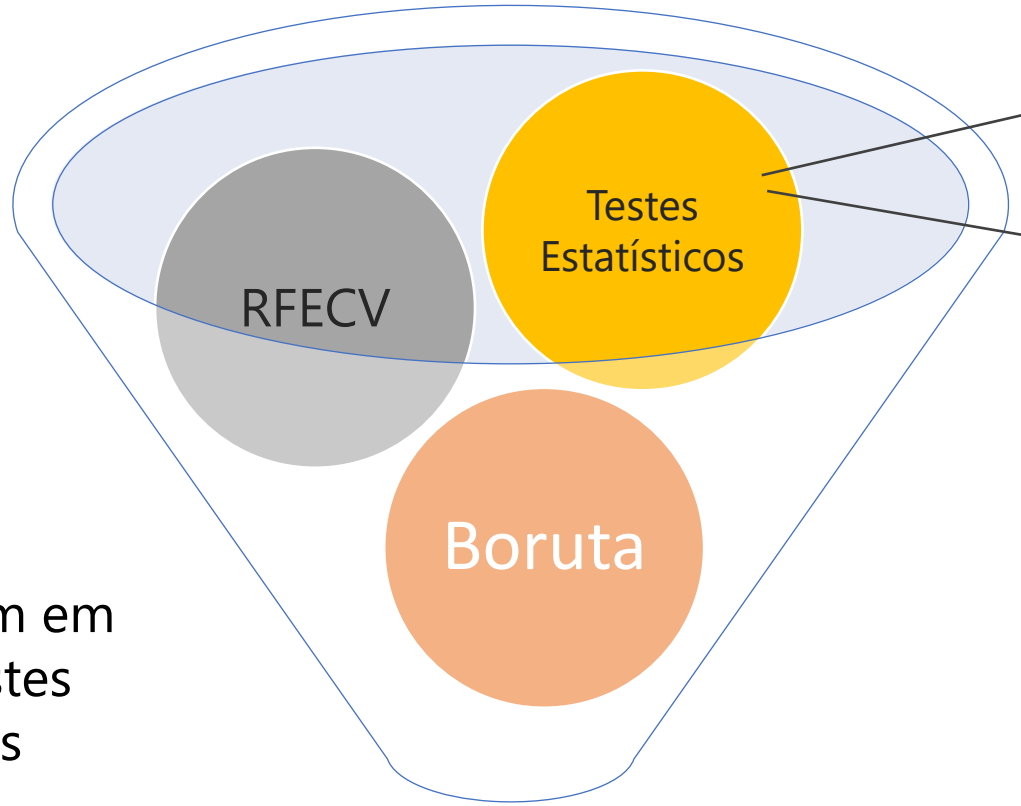
É feriado na semana

```
def eng_create_holiday_week(df , df_feriados):
    """
    Create new column called Semana de feriado.
    df = Dataframe
    df_feriados = Dafaframe contendo uma lista de feriados nacionais

    """
    #import da tabela feriado
    df_feriados = df_feriados.drop({'evento', 'status'}, axis=1)
    df_feriados.rename(columns={'data': 'Check-In' }, inplace=True)
    df_feriados['Check-In'] = pd.to_datetime(df_feriados['Check-In'], format ='%Y-%m-%d')
    df_feriados['Semana de Feriado'] = df_feriados['Check-In'].dt.week

    # Vamos juntar as duas tabelas Preço Médio e Feriados
    df = df.merge(df_feriados, left_on='Check-In', right_on='Check-In', how='left')
    #preenche os nulos com 0
    df = df.fillna(int(0))

    return df
```

```
df_train = eng_create_holiday_week(df_train, df_feriados)
```

Feature Selection

Processo de seleção das Features



Correlação (numéricas)

CHI-Quadrado (categóricas)

Features que passarem em pelo menos dois testes serão selecionadas

Teste de Multicolinearidade

Modelagem

Feature Selection

Teste Estatístico

```
['Número Hóspedes',
 'Número Quartos',
 'Número Camas',
 'Número Banheiros',
 'Avaliação',
 'Número Comentários'
 'Taxa',
 'Dia',
 'Mes']
```

```
['Localização',
 'Academia',
 'Ar-condicionado',
 'Cozinha',
 'Elevador',
 'Estacionamento gratuito',
 'Máquina de Lavar',
 'Permitido animais',
 'Piscina',
 'Secadora',
 'Self check-In',
 'Wi-Fi',
 'Café da Manhã',
 'Entrada/saída para esquis',
 'Jacuzzi',
 'Lareira interna',
 'Lava-louças',
 'Banheiro Compartilhado',
 'Vista',
 'Demanda',
 'É_feriado',
 'Semana_do_ano',
 'Semana de Feriado']
```

RFEVC

```
['Localização',
 'Número Hóspedes',
 'Número Quartos',
 'Número Camas',
 'Número Banheiros',
 'Avaliação',
 'Número Comentários',
 'Piscina',
 'Lava-louças',
 'Wi-Fi',
 'Café da Manhã',
 'Máquina de Lavar',
 'Estacionamento gratuito',
 'Academia',
 'Jacuzzi',
 'Secadora',
 'Entrada/saída para esquis',
 'Cozinha',
 'Permitido animais',
 'Lareira interna',
 'Ar-condicionado',
 'Self check-In',
 'Elevador',
 'Novo preço mais baixo',
 'Raridade',
 'Banheiro Compartilhado',
 'Taxa',
 'Vista',
 'Demanda',
 'É_feriado',
 'É outilier',
 'Mes',
 'Dia',
 'Semana_do_ano',
 'Semana de Feriado']
```

Boruta

```
['Localização',
 'Número Hóspedes',
 'Número Quartos',
 'Número Camas',
 'Número Banheiros',
 'Avaliação',
 'Número Comentários',
 'Piscina',
 'Academia',
 'Jacuzzi',
 'Secadora',
 'Permitido animais',
 'Taxa',
 'Mes',
 'Semana_do_ano']
```

Multicolinearidade Teste V-Cramer

	feature	VIF
7	Avaliação	24.386740
2	Número Quartos	18.434627
5	Mes	15.014006
3	Número Banheiros	14.024909
6	Número Hóspedes	12.705066
1	Número Camas	6.224634
4	Taxa	4.763461
0	Número Comentários	2.031651

	feature	VIF
5	Número Hóspedes	10.565242
2	Número Banheiros	8.419417
1	Número Camas	5.918743
4	Mes	5.793528
3	Taxa	4.267290
0	Número Comentários	1.933112

Código

Teste de Correlação

```python
# Teste de correlação
def teste_correlacao(df, Target, features, alpha=0.05):
    """
    df = Dataframe
    Target = target
    features = numeric features to be tested
    Return list of variable which are correlationed with target
    """
    correlation = []
    no_correlation = []
    for col in features:
        cor, p = stats.spearmanr(df_train[col],df_train[Target])
        if p <= alpha:
            correlation.append(col)
        else:
            no_correlation.append(col)

    return correlation
```

Teste QUI-Quadrado

```python
def teste_chi2_(df, Target='Target', features_cat=features_cat, aplha=0.05):
    """
    Chi2 test
    df = DataFrame
    Target = Target string
    features = List of categorical features
    aplha = indice of significance
    Return a list of variable which has passed for Chi2 test
    """
    from scipy.stats import chi2_contingency
    p_values_cat_features = {}
    for col in features_cat:
        # Cria tabela de contingencia
        df_cross = pd.crosstab(df[col], df[Target])
        # Aplica o teste e extrai o p-valor
        p_value = scipy.stats.chi2_contingency(df_cross)[1]
        # Armazena coluna e p-valor em um dict
        p_values_cat_features[col] = p_value

    p_values_cat_features = pd.Series(p_values_cat_features)
    filter_cat_features = p_values_cat_features[p_values_cat_features < aplha].index.tolist()

    return filter_cat_features
```

```python
chi2 = teste_chi2_(df_train, Target = 'Preço/Noite', features_cat=features_cat)
```

Código

Teste RFECV

```
def teste_rfe(X_train, y_train):
    """
    RFE Test
    X_train = Dataframe with all features to be tested and not target
    y_train = Target
    Return a list of varibales which has passed for the test
    """
    from sklearn.feature_selection import RFECV
    from sklearn.ensemble import RandomForestRegressor
    rfecv_RFC = RFECV(estimator=RandomForestRegressor(n_jobs = -1,
                                                      max_depth = 5),
                      scoring='neg_mean_squared_error')
    rfecv_RFC.fit(X_train,y_train)
    mask_RFC = rfecv_RFC.support_
    cols_selected_RFE = X_train.columns.tolist()

    return cols_selected_RFE
```

```
rfe = teste_rfe(X_train_copy, y_train)
```

Boruta

```
def teste_boruta(X_train, y_train):
    """
    Boruta Test - Return features which was aproved
    X_train = Dataframe with all features to be tested and not target
    y_train = Target
    Return a list of varibales which has passed for the test
    """
    from boruta import BorutaPy
    from sklearn.ensemble import RandomForestRegressor
    # -1 indica para o python usar todo o processador
    boruta_selector = BorutaPy(RandomForestRegressor(n_jobs = -1, max_depth = 5),
                    n_estimators = 50, max_iter=100, random_state = 0)
    boruta_selector.fit(np.array(X_train), np.array(y_train))
    boruta_selector.support_
    boruta_selector.support_weak_
    features_boruta = X_train.loc[:, boruta_selector.support_].columns.tolist()

    return features_boruta
```

```
boruta = teste_boruta(X_train_copy, y_train)
```

Código

Features que passaram pelos testes

```
def feature_results(correlation, chi2, rfe, boruta):
    """
    Return a list which has passed for three testes
    correlation = set of result from corrrelation teste
    chi2 = set of features which has passed from chi2 teste
    rfe = set of features which has passed from rfe teste
    boruta = set of features which has passed from boruta teste
    """
    # Junção dos testes de correlação e Chi2
    statistic_test = correlation + chi2
    # Interserção entre testes estatísticos e RFE
    statistic_test_rfe = list(set(statistic_test).intersection(set(rfe)))
    # Interserção entre testes estatísticos e boruta
    statistic_test_boruta = list(set(statistic_test).intersection(set(boruta)))
    # Interserção entre rfe e boruta
    rfe_boruta = list(set(rfe).intersection(set(boruta)))

    # Quem passou no primeiro que também passou no boruta
    feature_selection = set(statistic_test_rfe) &
                        set(statistic_test_boruta) &
                        set(rfe_boruta)

    return feature_selection
```

```
feature_results(correlation, chi2, rfe, boruta)
```

Teste de Multicolinearidade

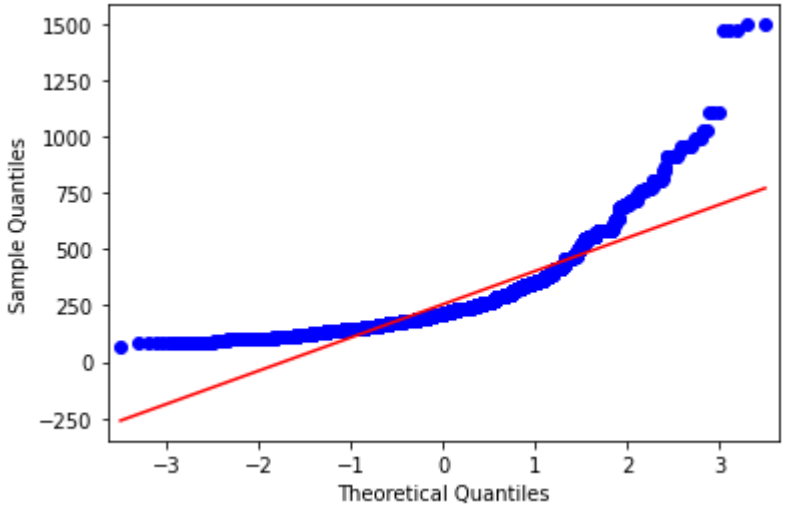
```
def multicolinearidade(df, features):
    """
    V-Cramer test
    X = DataFrame without target
    features = numerical features to be tested
    Return a Dataframe with VIF score
    """
    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = df.loc[:,features].columns
    # calculando VIF
    vif_data["VIF"] = [variance_inflation_factor(df.loc[:,features].values, i)
                       for i in range(len(df.loc[:,features].columns))]

    return vif_data.sort_values(by='VIF', ascending=False)
```

```
multicolinearidade(X_train, feature_selection_num)
```

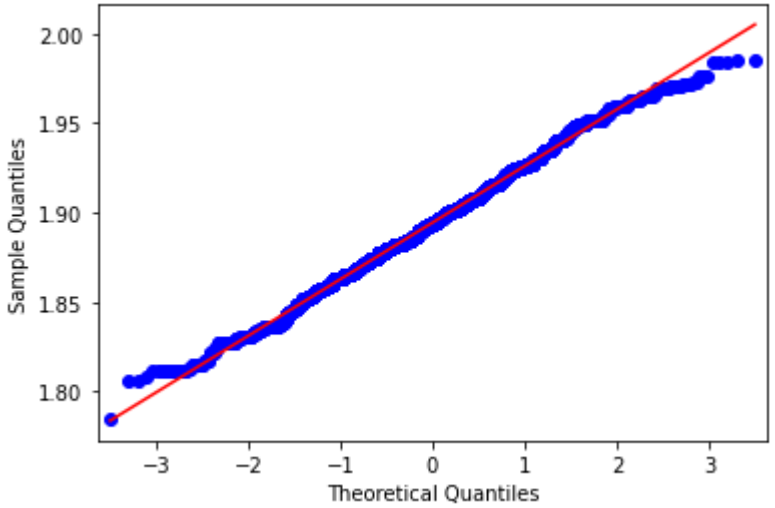
Data preparation

Umas das premissas da Regressão é que o Target precisa seguir uma curva normal e para isso precisamos testar a normalidade dele. O método gráfico do QQ-Plot e o teste estatístico do Shapiro Wilk indicam se é necessária uma transformação.

Antes



Depois



É possível verificar que os dados não são uma normal.

Agora os dados estão mais próximos de uma normal.

Código

QQ-Plot

```
y = df_train['Preço/Noite']
# qq plot
qqplot(y, line='s')
plt.show()
```

Teste Shapiro Wilk

```
def shapiro_test(df, alpha=0.05):
    """
    Shapiro Wilk test
    df = Dataframe
    alpha = indice of significancy. Default is 0.05
    """
    stat, p = scipy.stats.shapiro(df)
    print('p-valor: p=%.3f' % (p))
    # interpret
    alpha = alpha
    if p > alpha:
        print('Parece ser normalmente distribuiída. Não rejeitamos H0')
    else:
        print('NÃO parece ser normalmente distribuiída. Rejeitamos H0')
    return stat, p
```

```
shapiro_test(df_train)
```

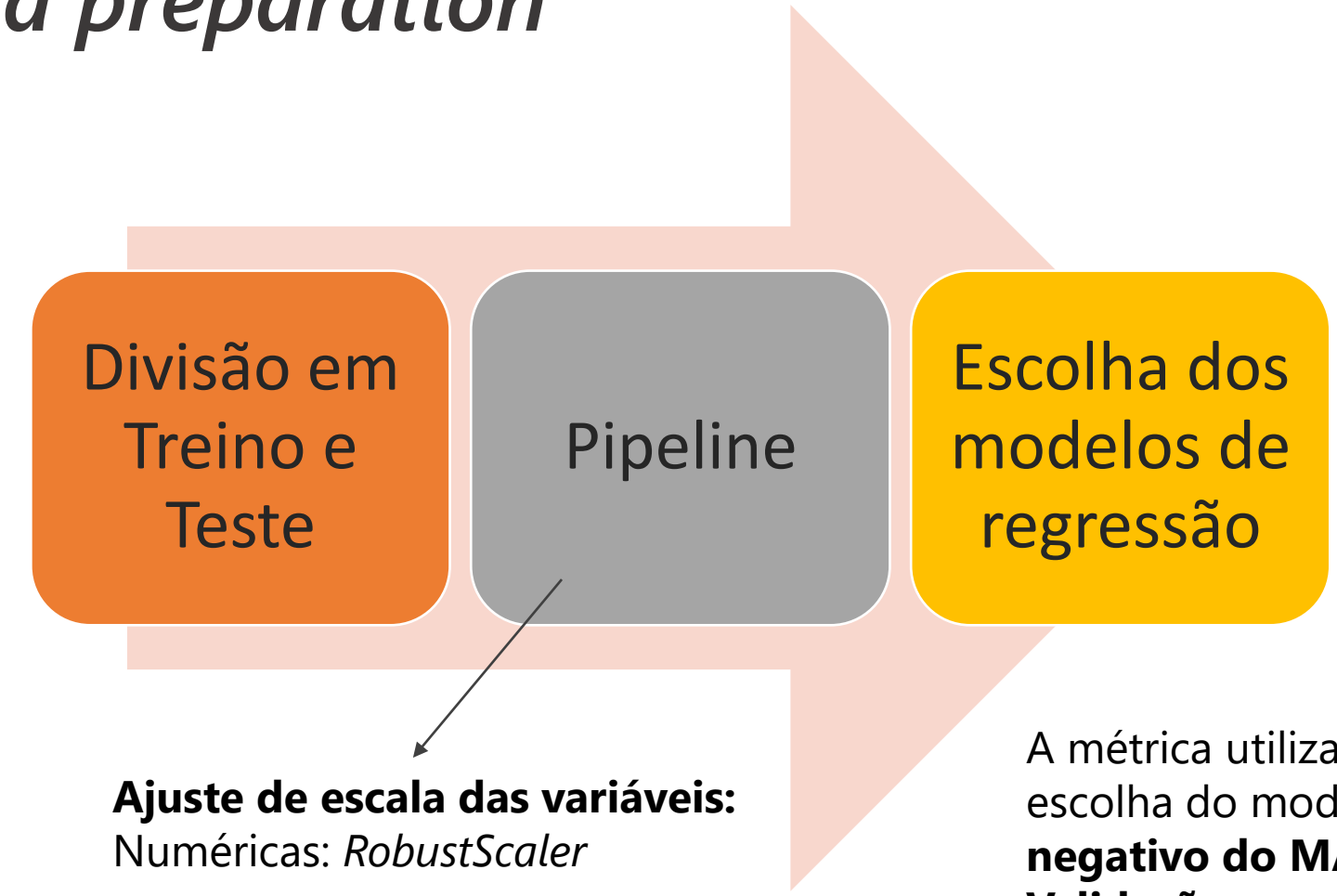
Transformação *Box Cos*

```
transformed_y, lambda_found = scipy.stats.boxcox(y)

print(f'Melhor lambda: {lambda_found}')
```

```
Melhor lambda: -0.4897796628352117
```

```
# qq plot
qqplot(transformed_y, line='s')
plt.show()
```

Data preparation



Ajuste de escala das variáveis:
Numéricas: *RobustScaler*

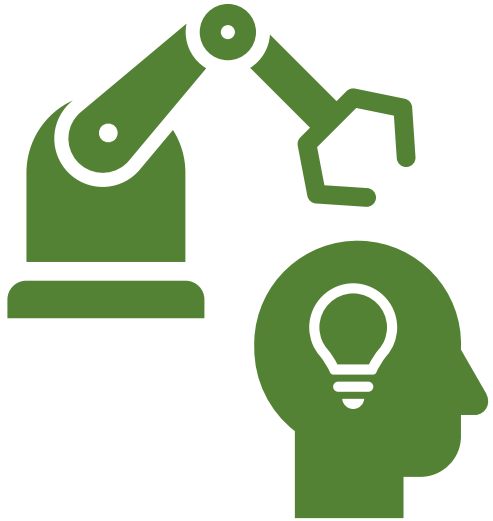
Categóricas: *FrequencyEnconder*

A métrica utilizada para escolha do modelo será o **negativo do MAE** com **Validação cruzada** 5 folds.

Os três modelos com a métrica mais próximas de zero serão escolhidos.

O que eu quero resolver?

Neste projeto, queremos resolver um problema de regressão. Assim, temos que encontrar a melhor equação que correlacione as variáveis (características do imóvel) que expliquem o preço/noite de aluguel do imóvel do Airbnb.



Modelos de Machine Learning

Regressão Linear

Decision Tree

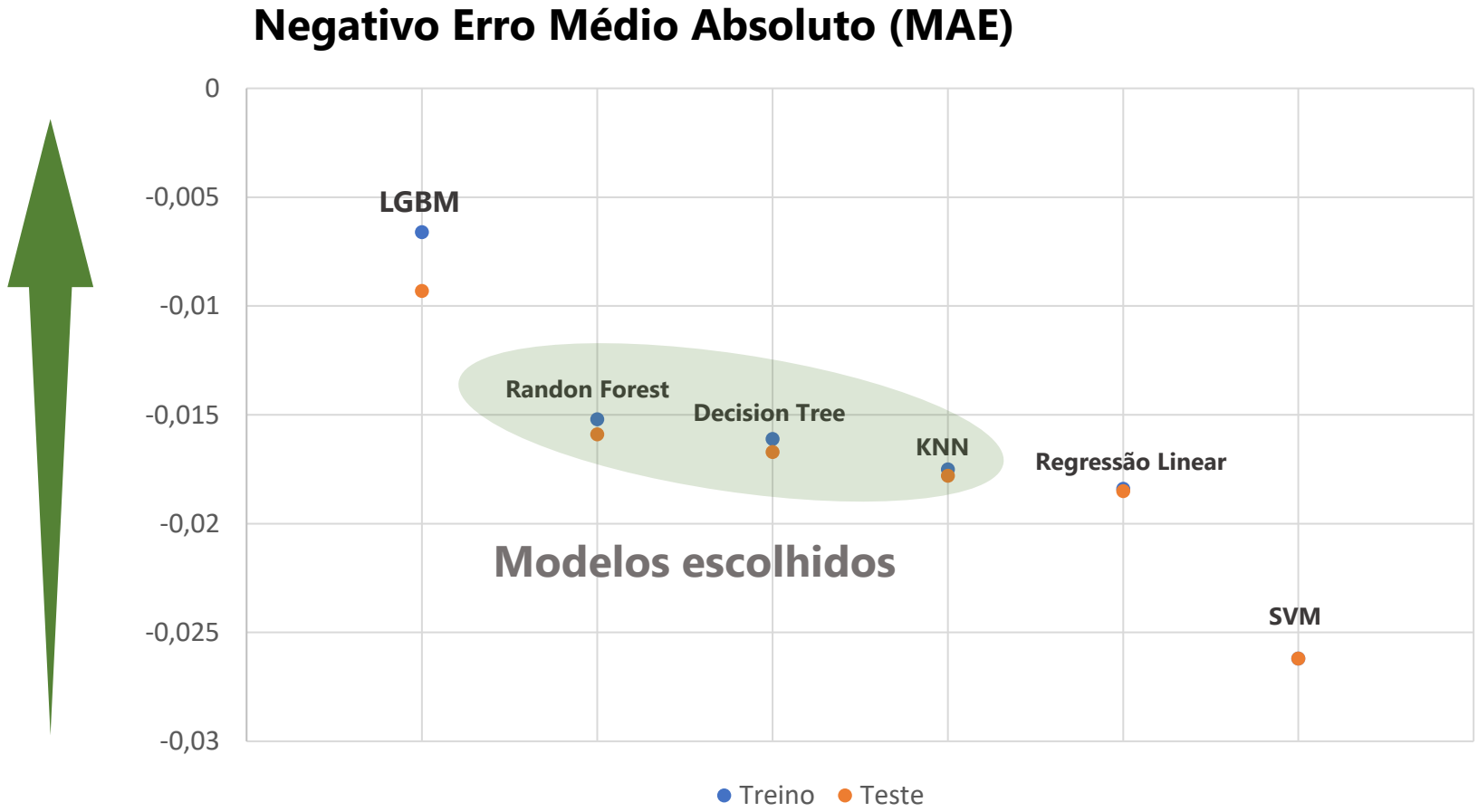
Random Forest

LGBM Regressor

SVM

KNN

Escolha dos modelos



Sendo uma métrica de erro, quando mais próxima de zero melhor.

* Embora LGBM Regressor tenha resultado em uma ótima performance é possível observar Overfitting.

Código

Pipeline

```
def create_pipe(model, pre_processor):
    """
    model = Model funccion
    Return a pipeline with model and transformations
    """
    pipe_num = Pipeline(steps=[('scaler', RobustScaler())])
    pipe_cat = Pipeline(steps=[('cfce', CountFrequencyEncoder(
                encoding_method='frequency', variables=['Localização']))])
    transformers = [pipe_num, pipe_cat]
    pre_processor = ColumnTransformer(transformers)
    pipe = Pipeline(steps=[('pre_processor', pre_processor),
                           ('model', model)])
    return pipe
```

Escolha dos modelos – *Cross Validation*

```
def cv_scores(estimator, X, y, cv=5, scoring='neg_mean_absolute_error'):
    """
    Cros Validation
    estimatior = create_pipe funccion with model and transformation
    X = X
    y = target
    scoring = score of model. Default is neg_mean_absolute_error for Regression Models
    cv = Default is 5 fold. Stratified K Fold is also possible.
    Print the score from train and test of modeling
    """
    cv_scores = cross_validate(estimator=estimator,
                               X = X,
                               y = y,
                               scoring =scoring,
                               cv = cv,
                               return_train_score=True)
    print(f'{scoring} no Treino: {np.mean(cv_scores["train_score"]):.4f}')
    print(f'{scoring} no Teste: {np.mean(cv_scores["test_score"]):.4f}')
```

Aplicação das funções

```
pipe_LR = create_pipe(LinearRegression(), pre_processor)
```

```
cv_scores(pipe_LR, X_train, y_train, scoring='neg_mean_absolute_error')
```

Otimização dos modelos

Grid de hiperparâmetros

KNN Regressor	Decision Tree	Random Forest
n_neighbors: range(1, 100), *weights:* ['uniform', 'distance'], *p:* range(1,11), *metric:* ['euclidean', 'manhattan', 'chebyshev', 'minkowski', 'wminkowski', 'seuclidean', 'mahalanobis']	*criterion:* ['mae'], *max_depth:* range(1, 11), *min_samples_split:* range(2, 41, 5), *min_samples_leaf:* range(2, 21, 2),	*n_estimators:* [10, 25, 50, 75, 100, 150], *criterion:* ['mae'], *max_depth:* range(1, 11), *min_samples_split:* range(2, 41, 5), *min_samples_leaf:* range(2, 21, 2), *max_features:* ['auto', 'sqrt', 'log2'],
Overfitting. Números de vizinhos resultou em 1, ou seja, ele mesmo.	**Scores: MAE** Treino: 0.0114 Teste: 0.0113	**Scores: MAE** Treino: 0.0107 Teste:0.0099

Código

Grid Search

```
param_grid_RF = {
                'model__n_estimators': [10, 25, 50, 75, 100, 150],
                'model__criterion' : ['mae'],
                'model__max_depth' : range(1, 11),
                'model__min_samples_split' : range(2, 41, 5),
                'model__min_samples_leaf' : range(2, 21, 2),
                'model__max_features': ['auto', 'sqrt', 'log2'],
                }
```

```
random_search_RF = RandomizedSearchCV(estimator=pipe_RF,
                                      param_distributions=param_grid_RF,
                                      scoring='neg_mean_absolute_error',
                                      random_state=123,
                                      cv=5,
                                      n_jobs=-1,
                                      verbose=1,
                                      n_iter=250)
```

```
random_search_RF.fit(X_train_, y_train_)
```

```
Fitting 5 folds for each of 250 candidates, totalling 1250 fits
```

Modelagem

Random Forest Regressor

.fit

.predict

```
best_RF.fit(X_train_, y_train_)
```

```
Pipeline(steps=[('scaler', RobustScaler()),
                ('model',
                 RandomForestRegressor(criterion='mae', max_depth=10,
                                       min_samples_leaf=2, min_samples_split=7,
                                       random_state=123))])
```

Melhores parâmetros

```
result_RF = best_RF.predict(X_test_)
```

```
diaria_RF = scipy.special.inv_boxcox(result_RF, lambda_found)
```

Inversa BoxCos

```
np.round(diaria_RF, 2)
```

```
array([182.99, 156.62, 186.74, 233.1 , 210.69, 318.06, 324.34, 204.67,
       221.8 , 162.33, 183.75, 248.75, 176.51, 160.72, 173.05, 576.65,
       144.09, 217.85, 191.  , 194.84, 169.71, 175.54, 195.89, 185.37,
       231.16, 216.73, 185.06, 202.39, 190.41, 465.21, 320.84, 241.34,
```

Predições

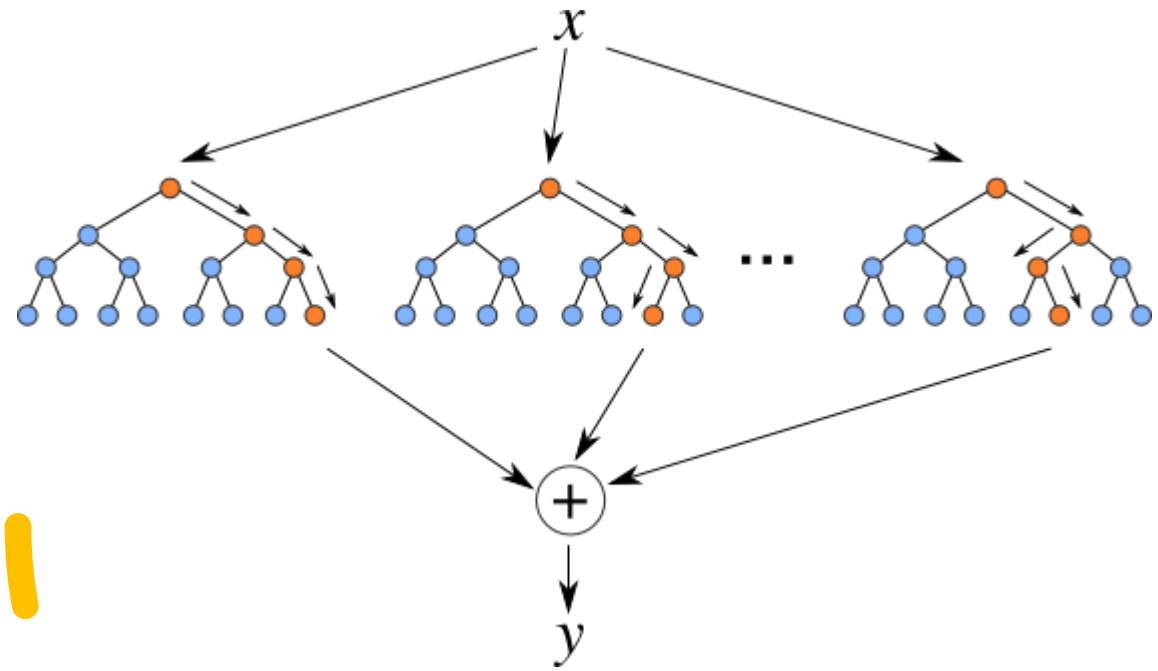
```
MODEL_OUTPUT_PATH = '../models'
MODEL_OUPUT_NAME = 'model_v0.pkl'
```

```
pickle.dump(best_RF, open(os.path.join(MODEL_OUTPUT_PATH, MODEL_OUPUT_NAME ), 'wb'))
```

Salva a predição em *.pickle*

Performance

Random Forest



Erro Médio Quadrático (MSE)
6474.15

Erro Médio Absoluto (MAE)
50,45

% Erro Médio Absoluto (MAPE)
19,13%

Para modelos de precificação considerando base interna a performance está em torno de 5% e 10%.

Todavia, considerando uma base pequena com 4209 observações obteve uma ótima performance.

Código

Métricas

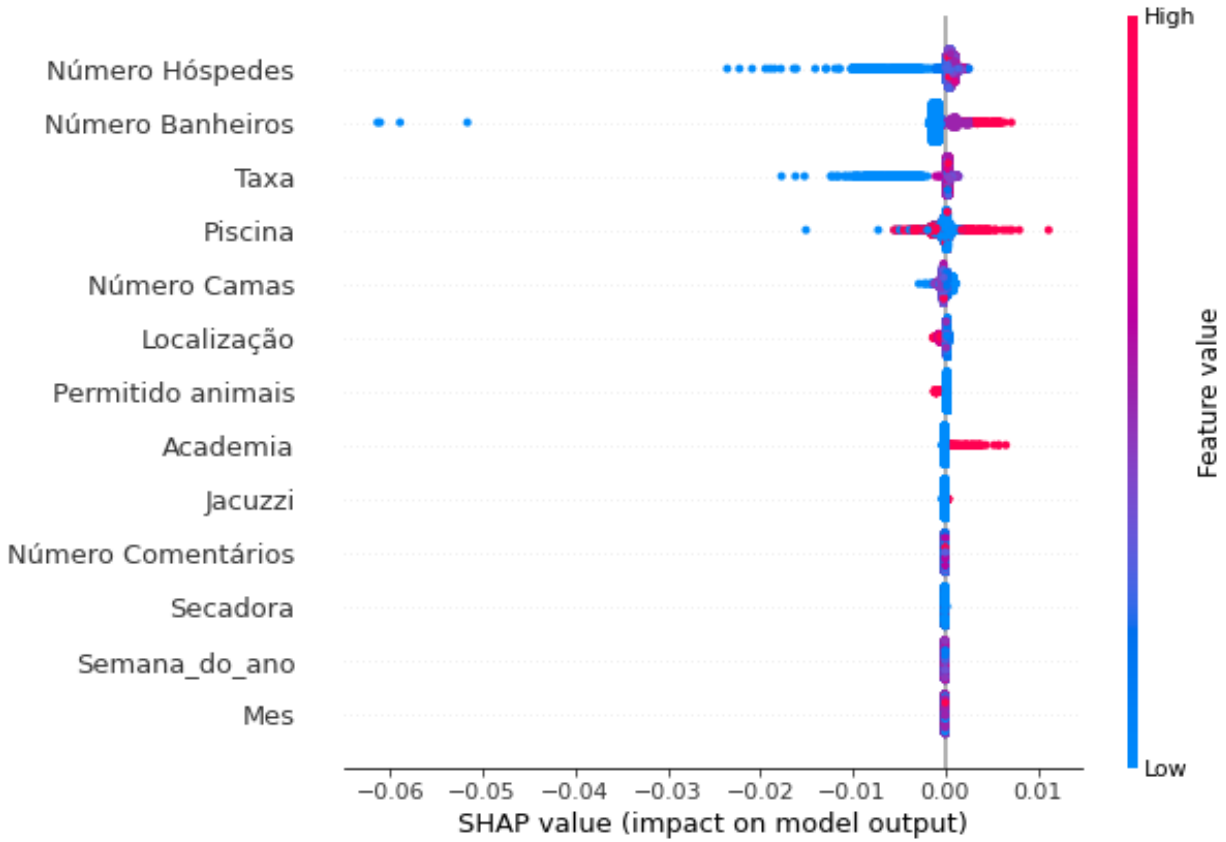
```
# Erro médio Quadratico
mse = np.mean((y__test - previsao_preco_noite)**2)
print(f'MSE: {round(mse, 2)}')
```

```
# Erro Médio Absoluto
mae = np.mean(abs(y__test - previsao_preco_noite))
print(f'MAE: {round(mae, 2)}')
```

```
# Porcentagem do Erro médio Absoluto
mape = 100*np.mean(abs(y__test - previsao_preco_noite)/ y__test)
print(f'MAPE: {round(mape, 2)}')
```

Interpretação do modelo

Random Forest Regressor

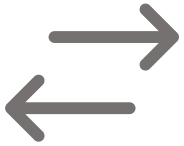



```
import shap
X_importance = df_train.drop('Preço/Noite', axis=1)
explainer = shap.Explainer(best_RF.predict, X_importance)
shap_values = explainer(X_importance)
shap.summary_plot(shap_values, X_importance)
```

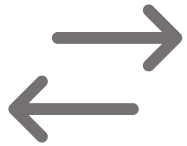
Utilizando a biblioteca SHAP para análise.

Podemos observar as features que mais explicam o modelo.

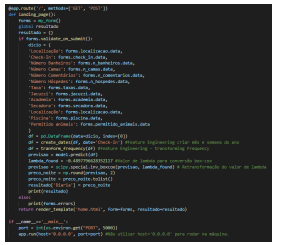
Deploy



HTML



build.py



Flask python

Neste projeto não houve a criação de um banco de dados. Então os dados informados pelo usuário são direcionados a uma classe de formulário.

- Definição de classe do formulário
- Modelo .pickle
- Estância de um objeto framework Flask

Deploy

build.py

Modelo treinado

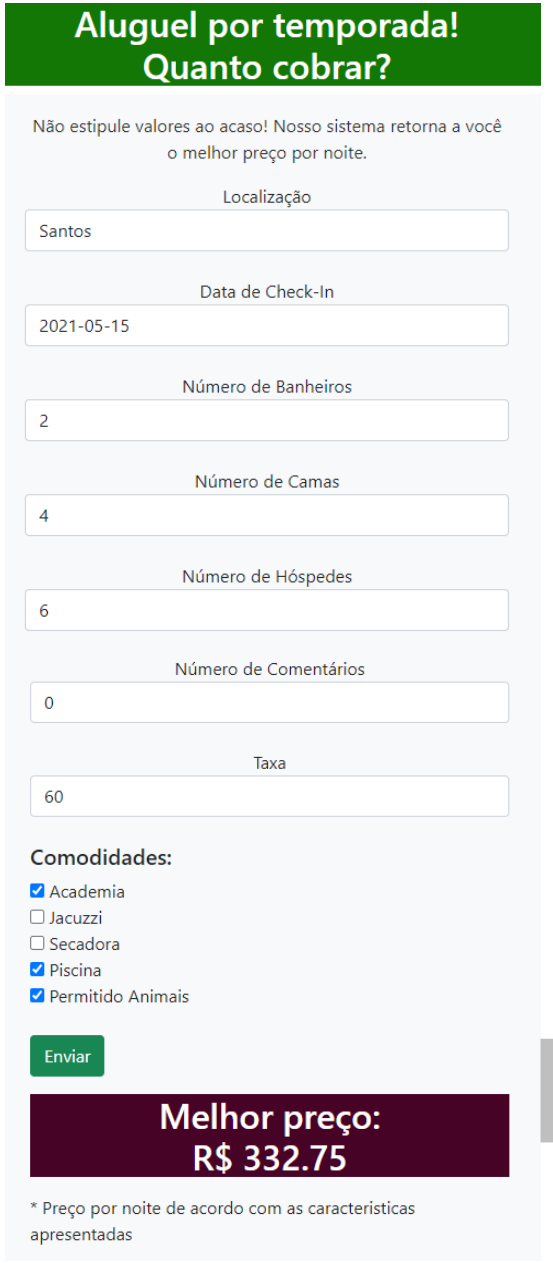
```python
# Import do modelo:
MODEL_NAME = 'model_v0.pkl'
MODEL_PATH = '../models'
model = pickle.load(open(os.path.join(MODEL_PATH, MODEL_NAME), 'rb'))
```

```python
#Instancia um objeto do flask
app = Flask(__name__, template_folder='template')
app.config['SECRET_KEY'] = 'minha_senha'

@app.route('/', methods=['GET', 'POST'])
def landing_page():
    forms = my_form()
    global resultado
    resultado = {}
    if forms.validate_on_submit():
        dicio = {
        'Localização': forms.localizacao.data,
        'Check-In': forms.check_in.data,
        'Número Banheiros': forms.n_banheiros.data,
        'Número Camas': forms.n_camas.data,
        'Número Comentários': forms.n_comentarios.data,
        'Número Hóspedes': forms.n_hospedes.data,
        'Taxa': forms.taxas.data,
        'Jacuzzi': forms.jacuzzi.data,
        'Academia': forms.academia.data,
        'Secadora': forms.secadora.data,
        'Localização': forms.localizacao.data,
        'Piscina': forms.piscina.data,
        'Permitido animais': forms.permitido_animais.data
        }
        df = pd.DataFrame(data=dicio, index=[0])
        df = create_dates(df, date='Check-In') #feature Engineering criar mês e semana do ano
        df = tranform_frequency(df) #feature Engineering - transforming Frequency
        previsao = model.predict(df)
        lambda_found = -0.4897796628352117 #Valor de lambda para conversão box-cox
        previsao = scipy.special.inv_boxcox(previsao, lambda_found) # Retransformação do valor de lambda
        preco_noite = np.round(previsao, 2)
        preco_noite = preco_noite.tolist()
        resultado['Diaria'] = preco_noite
        print(resultado)
    else:
        print(forms.errors)
    return render_template('home.html', form=forms, resultado=resultado)

if __name__=='__main__':
    port = int(os.environ.get("PORT", 5000))
    app.run(host='0.0.0.0', port=port) #Não utlizar host='0.0.0.0' para rodar na máquina.
```

Predição

Aplicação on-line

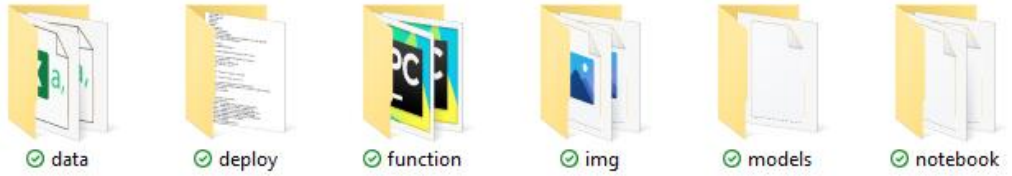
O usuário informa os itens e a aplicação retorna com o preço/noite.



Preço sugerido

Organização do Projeto

Pastas de projeto



Dados

- data_scrapping
- baixada_santista
- calendario
- DATA_baixada_santista_raw
- DATA_feature_selected
- DATA_feature_selected_V2
- DATA_TEST
- data_test_for_modeling
- DATA_train_cleaned
- DATA_train_eng
- feriados_nacionais_2021
- airbnb_bertioga_fev
- airbnb_bertioga_mar
- airbnb_guaruja_fev
- airbnb_guaruja_mar
- airbnb_itanhaem_fev
- airbnb_itanhaem_mar
- airbnb_monguagua_fev
- airbnb_monguagua_mar
- airbnb_peruibe_fev
- airbnb_peruibe_mar
- airbnb_praiagrande_fev
- airbnb_praiagrande_mar
- airbnb_santos_abril_
- airbnb_santos_fevereiro
- airbnb_santos_marco_
- airbnbsantos_extracao_2401

Deploy

- models
- template
- build
- model_v0.pkl
- Procfile
- requirements

Modelos

- model_v0.pkl

Funções

- __init__
- cleaning_errors_from_scrapping
- data_cleaning
- feature_eng
- feature_selection
- scrapping
- scrapping_cleaning

Notebooks

- 1-1 Coleta de Dados - Scraping Airbnb_Baixada_Santista_Full.ipynb
- 1-2 Coleta de Dados - Concate_All_Csv_By_Scrapping.ipynb
- 2-1 Limpeza de Dados - Primeira Limpeza.ipynb
- 2-2 Limpeza de Dados - Segunda Limpeza.ipynb
- 2-3 Limpeza de Dados - Terceira Limpeza.ipynb
- 3 - EDA_feat_engineering.ipynb
- 4 Feature_selection.ipynb
- 5. Modelagem e Métricas de Performance.ipynb
- Processamento_dados de Teste.ipynb

Bibliotecas

```python
import pandas as pd
import numpy as np

# Scrapping
from bs4 import BeautifulSoup
import requests

# Visualização
import matplotlib.pyplot as plt
import seaborn as sns

# Testes Estatísticos
import scipy
from scipy.stats import stats
import statsmodels.api as sm
from statsmodels.graphics.gofplots import qqplot

# Data Preparation
from feature_engine.encoding import CountFrequencyEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Cluster
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from kneed import KneeLocator
```

```python
# Feature selection:
from sklearn.feature_selection import f_classif, chi2
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import chi2_contingency
from scipy.stats import pointbiserialr
from boruta import BorutaPy
from sklearn.feature_selection import RFECV

# Cross Validation
from sklearn.model_selection import train_test_split, cross_validate, RandomizedSearchCV
from sklearn.model_selection import cross_val_score

# Modelos de Regressão
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor

# Métricas
from sklearn.metrics import log_loss

# Feature Importance
import shap

# Deploy do modelo
import pickle

import warnings
warnings.filterwarnings("ignore")
```

Autor do Projeto

Luiz Alves

Data Science pela Digital House.
Pós-graduado em Marketing e Propaganda e Graduado em Desenho Industrial – Design pela Universidade São Judas Tadeu (USJT).

11 99234-6616

luizn.mkt@gmail.com

linkedin.com/in/luizn

GuitHub

github.com/luizdatamkt

Digital House > Coding School

Projeto elaborado durante o curso de Data Science na Digital House.
Supervisão dos professores Bruno Viera e Maurício Nascimento.

Participação dos colegas de turma:
Caio Akira, Ivan Russeto e Vitor Bonde.