In [5]:
```
# read the hotel reservation file
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import umap
from scipy import stats
import plotly.express as px
from sklearn.manifold import TSNE
hotel = pd.read_csv('Hotel Reservations.csv')
```

In [49]:
```
hotel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 18 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   no_of_adults                          36275 non-null  int64  
 1   no_of_children                        36275 non-null  int64  
 2   no_of_weekend_nights                  36275 non-null  int64  
 3   no_of_week_nights                     36275 non-null  int64  
 4   type_of_meal_plan                     36275 non-null  object 
 5   required_car_parking_space            36275 non-null  int64  
 6   room_type_reserved                    36275 non-null  object 
 7   lead_time                             36275 non-null  int64  
 8   arrival_year                          36275 non-null  int64  
 9   arrival_month                         36275 non-null  int64  
 10  arrival_date                          36275 non-null  int64  
 11  market_segment_type                   36275 non-null  object 
 12  repeated_guest                        36275 non-null  int64  
 13  no_of_previous_cancellations          36275 non-null  int64  
 14  no_of_previous_bookings_not_canceled  36275 non-null  int64  
 15  avg_price_per_room                    36275 non-null  float64
 16  no_of_special_requests                36275 non-null  int64  
 17  booking_status                        36275 non-null  object 
dtypes: float64(1), int64(13), object(4)
memory usage: 5.0+ MB
```

In [6]:
```
categorical_columns = [
    "type_of_meal_plan",
    "required_car_parking_space",
    "room_type_reserved",
    "market_segment_type",
    "repeated_guest",
    "booking_status"
]

hotel = hotel.drop(['Booking_ID'], axis=1)     # Booking_ID is useless

numerical_columns = hotel.columns.difference(categorical_columns)
```

Preprocessing

Delete rows that normally would not make sense, such as :

- reservation without any adults (139 rows)

In [7]:
```
# check for null number
print(hotel.isnull().sum())
print("Empty room reservations : ", len(hotel[(hotel["no_of_adults"]==0) & (hotel["no_of
```

```
hotel.drop(hotel[(hotel["no_of_adults"]==0) & (hotel["no_of_children"]>0)].index,axis=0,
```

```
no_of_adults                            0
no_of_children                          0
no_of_weekend_nights                    0
no_of_week_nights                       0
type_of_meal_plan                       0
required_car_parking_space              0
room_type_reserved                      0
lead_time                               0
arrival_year                            0
arrival_month                           0
arrival_date                            0
market_segment_type                     0
repeated_guest                          0
no_of_previous_cancellations            0
no_of_previous_bookings_not_canceled    0
avg_price_per_room                      0
no_of_special_requests                  0
booking_status                          0
dtype: int64
Empty room reservations :  139
```

Central tendency of numerical variables

In [6]:
```
hotel[numerical_columns].describe()     # check for outliers dsa
```

Out[6]:

	arrival_date	arrival_month	arrival_year	avg_price_per_room	lead_time	no_of_adults	no_of_children
count	36136.000000	36136.000000	36136.000000	36136.000000	36136.000000	36136.000000	36136.000000
mean	15.589883	7.424424	2017.820013	103.507653	85.182090	1.852059	0.097880
std	8.740466	3.068408	0.384182	35.061640	85.951426	0.506908	0.385097
min	1.000000	1.000000	2017.000000	0.000000	0.000000	1.000000	0.000000
25%	8.000000	5.000000	2018.000000	80.375000	17.000000	2.000000	0.000000
50%	16.000000	8.000000	2018.000000	99.455000	57.000000	2.000000	0.000000
75%	23.000000	10.000000	2018.000000	120.120000	126.000000	2.000000	0.000000
max	31.000000	12.000000	2018.000000	540.000000	443.000000	4.000000	10.000000

Spreading/distribution

Skeweness

- Skewness = 0 (normally distributed)
- Skewness > 0 (positively skewed. Longer tail on the right side of the distribution)
  In positively skewed, the mean of the data is greater than the median (a large number of data-pushed on the right-hand side). A high level of skewness can cause misleading results.
- Skewness < 0 (negatively skewed. Longer tail on the left side of the distribution)
  In negatively skewed, the mean of the data is less than the median (a large number of data-pushed on the left-hand side).

Kurtosis

- Excess Kurtosis = Kurtosis - 3 (0 for normal distribution)
- Leptokurtic: Kurtosis > 0 (more outliers than normal distribution)
- Platykurtic: Kurtosis < 0 (fewer outliers than normal distribution)

In [7]:
```
# imprastierea variabilelor numerice
def dispersion(col):
    return [col.max() - col.min(), col.var(), col.std(), col.skew(), col.kurt()]

dispersion_table = pd.DataFrame(columns=['Range', 'Variance', 'Standard Deviation', 'Ske
for column in numerical_columns:
    dispersion_table.loc[column] = dispersion(hotel[column])

dispersion_table
```

Out[7]:

	Range	Variance	Standard Deviation	Skewness	Kurtosis
arrival_date	30.0	76.395740	8.740466	0.029624	-1.156948
arrival_month	11.0	9.415127	3.068408	-0.348185	-0.933292
arrival_year	1.0	0.147596	0.384182	-1.666040	0.775731
avg_price_per_room	540.0	1229.318596	35.061640	0.676135	3.158032
lead_time	443.0	7387.647595	85.951426	1.294455	1.184577
no_of_adults	3.0	0.256956	0.506908	-0.215035	0.534689
no_of_children	10.0	0.148299	0.385097	4.965036	42.747885
no_of_previous_bookings_not_canceled	58.0	3.088861	1.757515	19.213120	455.617403
no_of_previous_cancellations	13.0	0.136188	0.369036	25.151407	729.915012
no_of_special_requests	5.0	0.617111	0.785564	1.145437	0.877823
no_of_week_nights	17.0	1.990457	1.410836	1.598537	7.800162
no_of_weekend_nights	7.0	0.757803	0.870519	0.739832	0.307992

The big values for Kurtosis on columns: {'no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled'} are due to the fact that most of values for these variables are 0 and the other variables that appear could be interpreted as outliers. Also, a value for Skewness between -0.5 and 0.5 is symmetrical and we can see that on some columns: {'no_of_adults', 'arrival_month', 'arrival_date' }

Frequencies of categorical attributes

In [8]:
```
freq_table = pd.DataFrame(columns=['Value','Frequency', 'Percentage'])
for column in categorical_columns:
    freq_table.loc[column] = [hotel[column].value_counts().index[0], hotel[column].value

freq_table
```

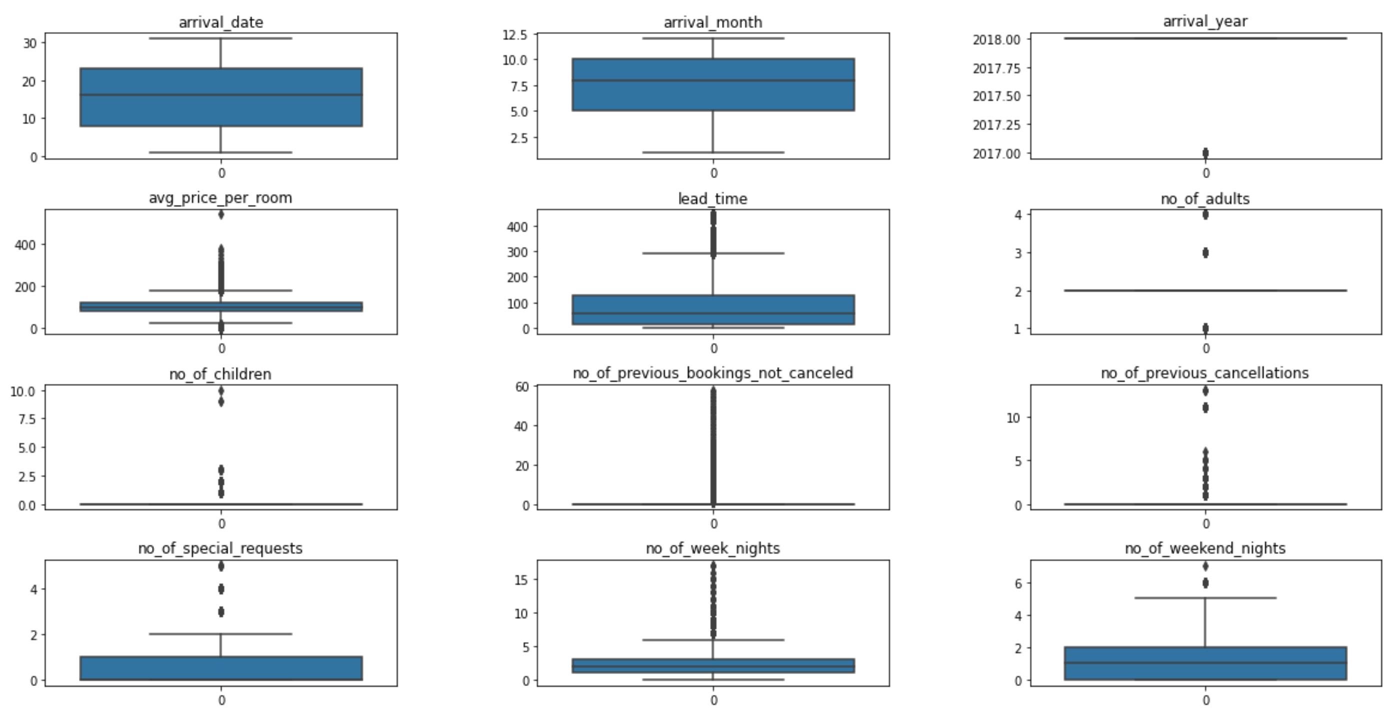
Out[8]:

	Value	Frequency	Percentage
type_of_meal_plan	Meal Plan 1	27698	76.649325
required_car_parking_space	0.0	35013.0	96.892296
room_type_reserved	Room_Type 1	28127	77.836507

market_segment_type	Online	23080	63.869825
repeated_guest	0.0	35206.0	97.426389
booking_status	Not_Canceled	24295	67.232123

In [13]:

```
# graphs (box plot)
def box_plot(col):
    plt.boxplot(col)
    plt.title(col.name)
    plt.show()

fig = plt.figure(figsize=(20, 10))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i, col in enumerate(numerical_columns):
    fig.add_subplot(4, 3, i+1)
    sns.boxplot(hotel[col])
    plt.title(col)
```



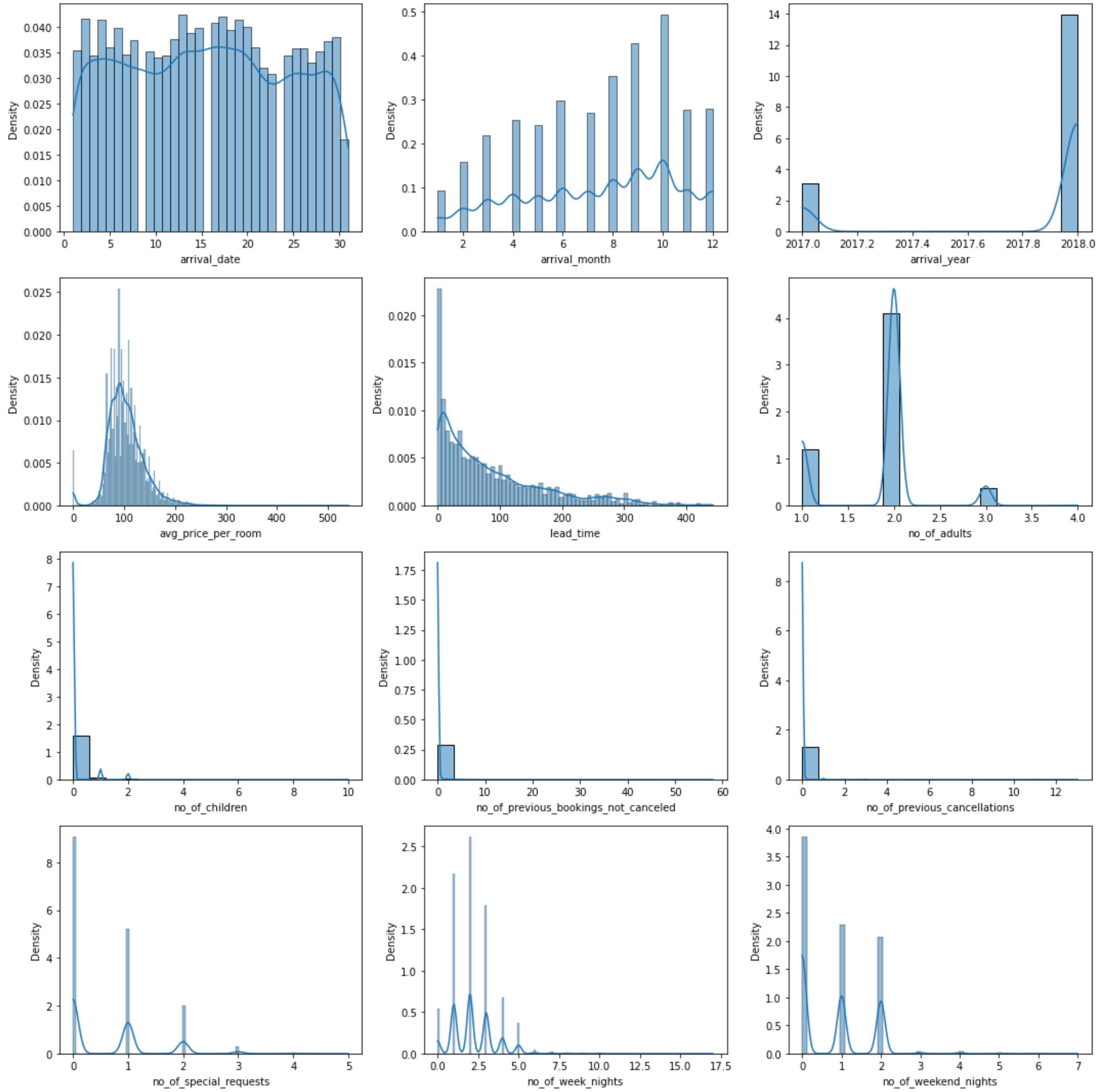
We can observe that for {no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled', 'no_of_children'} we don't see the box because of too many 0 values: so the min, Q1, median and also the Q3 are equal to 0.

Histograms for numeric attributes

In [14]:

```
figure = plt.figure()
figure.set_size_inches(10, 10)
figure, axes = plt.subplots(nrows=4, ncols=3, figsize=(15, 15))
figure.subplots_adjust(hspace = 1, wspace = 0.4)

for i, ax in enumerate(axes.flat, start=1):
    # kde = True -> Add a kernel density estimate to smooth the histogram, providing com
    # stat = 'density' -> Normalize the histogram to form a probability density: normali
    sns.histplot(hotel[numerical_columns[i-1]], ax=ax, kde=True, stat='density')

# padding between subplots
plt.tight_layout()
plt.show()
```

<Figure size 720x720 with 0 Axes>

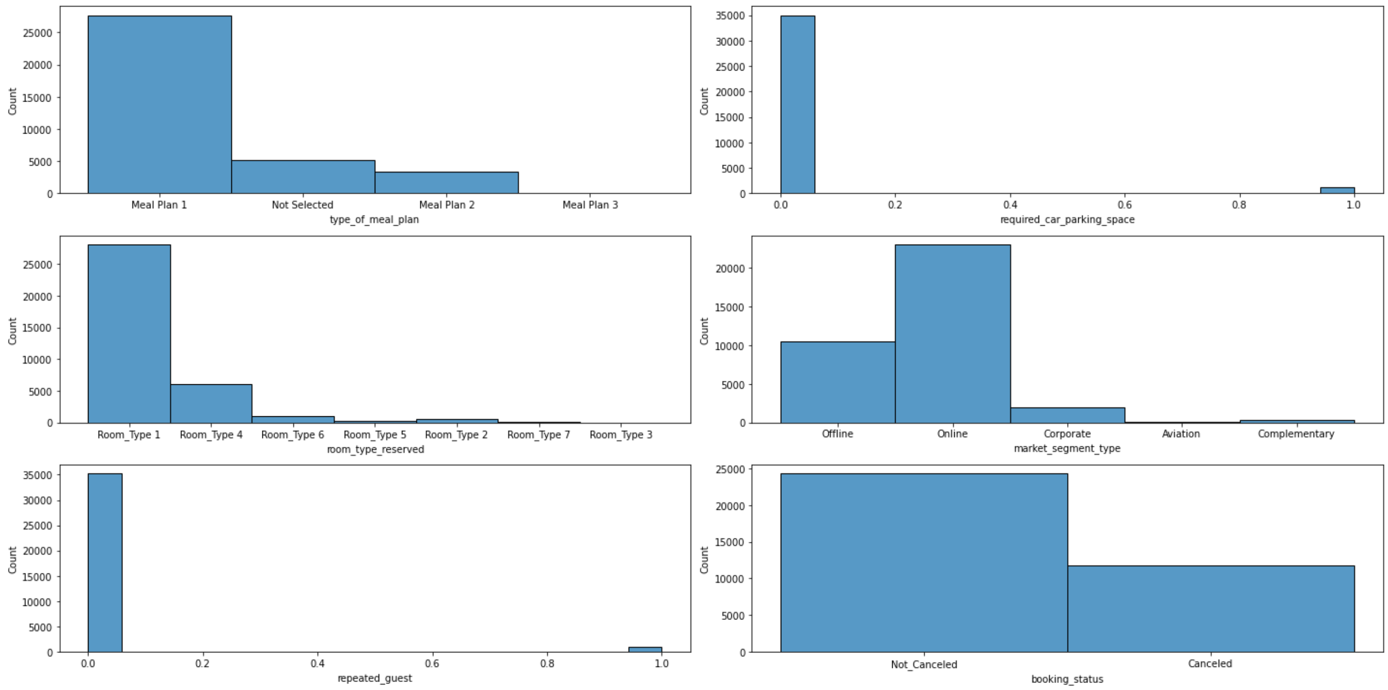


Histograms for categorical attributes

In [15]:

```
figure = plt.figure()
figure.set_size_inches(10, 10)
figure, axes = plt.subplots(nrows=3, ncols=2, figsize=(20, 10))
figure.subplots_adjust(hspace = 1, wspace = 0.4)

for i, ax in enumerate(axes.flat, start=1):
    sns.histplot(hotel[categorical_columns[i-1]], ax=ax)

plt.tight_layout()
plt.show()
```

<Figure size 720x720 with 0 Axes>



Bivariate analysis

Calculation of the correlation between numerical variables

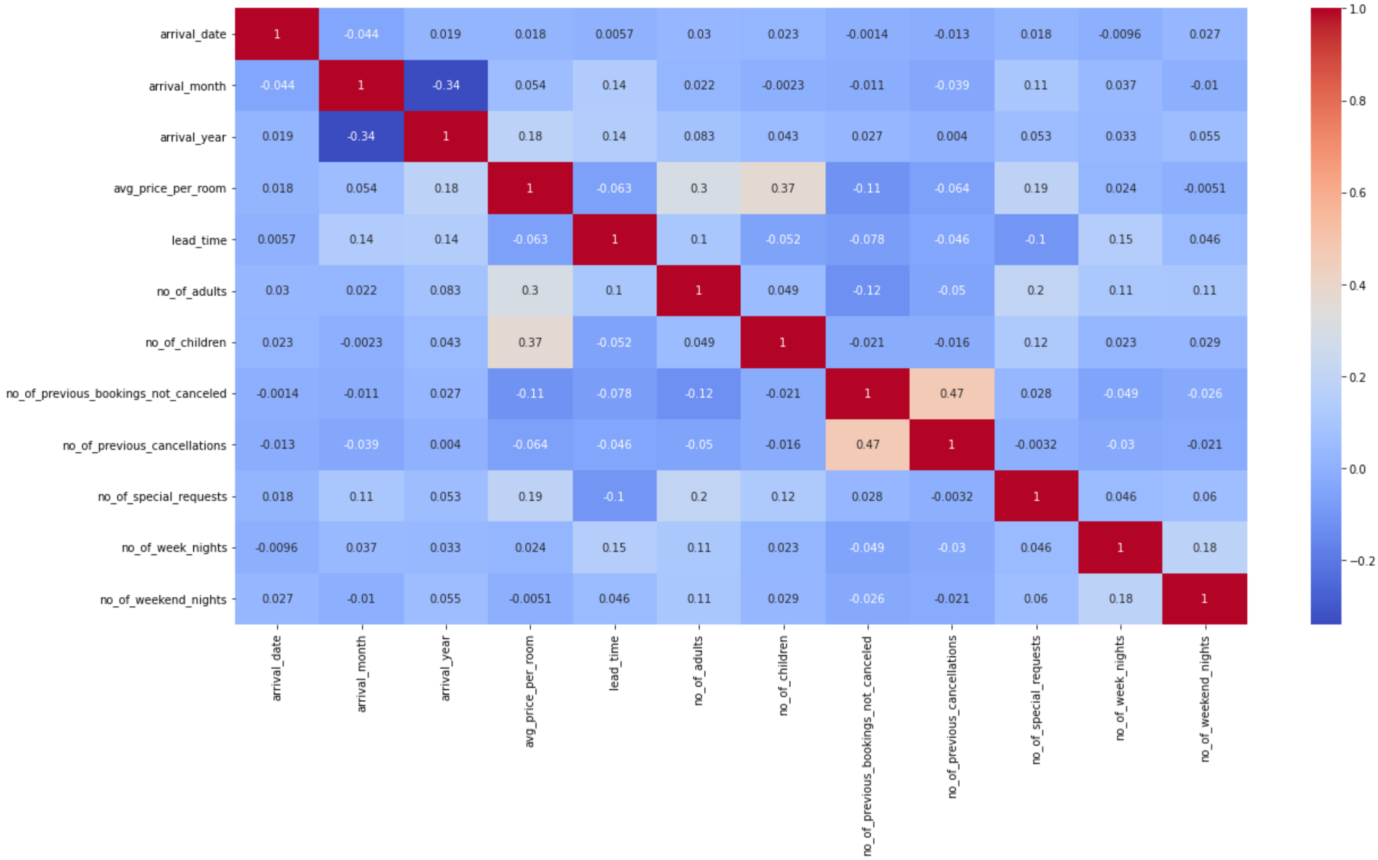
In [16]:
```
corr_matrix = hotel[numerical_columns].corr()
corr_matrix
```

Out[16]:

	arrival_date	arrival_month	arrival_year	avg_price_per_room	lead_time	no_
arrival_date	1.000000	-0.043546	0.018952	0.018237	0.005717	
arrival_month	-0.043546	1.000000	-0.339944	0.053747	0.136309	
arrival_year	0.018952	-0.339944	1.000000	0.179146	0.143256	
avg_price_per_room	0.018237	0.053747	0.179146	1.000000	-0.062617	
lead_time	0.005717	0.136309	0.143256	-0.062617	1.000000	
no_of_adults	0.030028	0.021528	0.082675	0.296445	0.102057	
no_of_children	0.022648	-0.002276	0.043137	0.366890	-0.052184	
no_of_previous_bookings_not_canceled	-0.001430	-0.010766	0.026544	-0.114204	-0.078218	
no_of_previous_cancellations	-0.012515	-0.038724	0.003991	-0.063665	-0.045763	
no_of_special_requests	0.017830	0.110130	0.052832	0.185859	-0.102255	
no_of_week_nights	-0.009551	0.037266	0.032583	0.023530	0.149263	
no_of_weekend_nights	0.027170	-0.010103	0.055111	-0.005090	0.046006	

In [17]:
```
# make the graphic wider
plt.figure(figsize=(20, 10))
# show value with a heatmap in order to see the correlation between the variables better
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```

Out[17]: <AxesSubplot:>



- We noticed a correlation between the number of adults and the average price of the room, between the number of children and the average price of the room (with the increase in the number of people, the price of the room also increases).

Teste de independenta

Chi-square test for categorical attributes

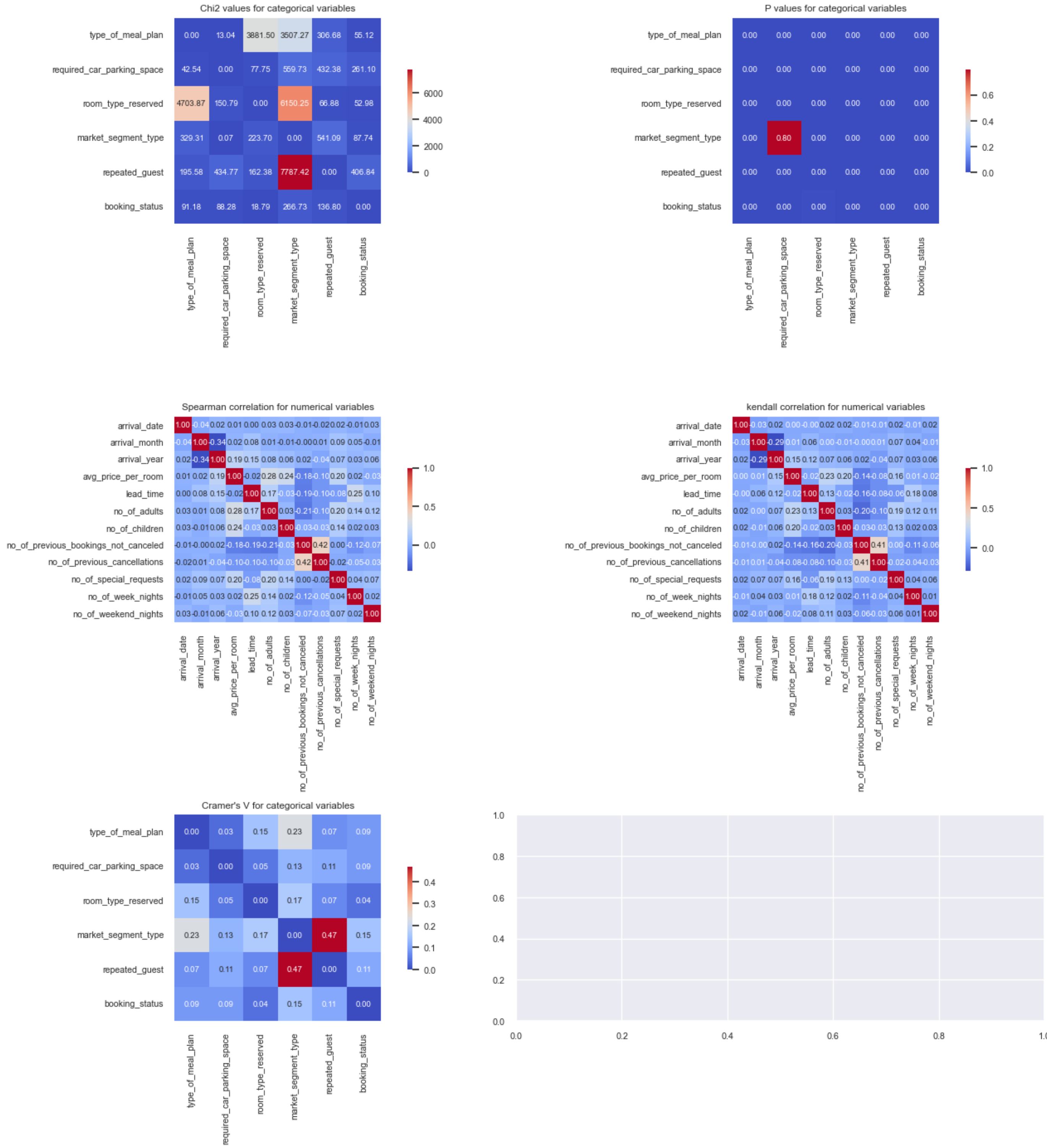
Like all hypothesis tests, the chi-square test of independence evaluates a null and alternative hypothesis. The hypotheses are two competing answers to the question “Are variable 1 and variable 2 related?”

```
Null hypothesis (H0): Variable 1 and variable 2 are not related in the
population; The proportions of variable 1 are the same for different values of
variable 2.
Alternative hypothesis (Ha): Variable 1 and  variable 2 are related in the
population; The proportions of variable 1 are not the same for different values
of  variable 2.
```

In [38]:
```
from sklearn.feature_selection import chi2
import numpy as np

resultant = pd.DataFrame(data=[(0 for _ in range(len(categorical_columns))) for _ in ran
                         columns=list(categorical_columns))
resultant.set_index(pd.Index(list(categorical_columns)), inplace = True)

resultant_chi = pd.DataFrame(data=[(0 for _ in range(len(categorical_columns))) for _ in
                         columns=list(categorical_columns))
resultant_chi.set_index(pd.Index(list(categorical_columns)), inplace = True)


for column in categorical_columns:
```

```
    # convert the categorical variable into dummy/indicator variables
    hotel[column] = pd.Categorical(hotel[column]).codes

for col1 in categorical_columns:
    for col2 in categorical_columns:
        if col1 != col2:
            chi2_val, p_val = chi2(np.array(hotel[col1]).reshape(-1, 1), np.array(hotel[
            resultant.loc[col1, col2] = p_val
            resultant_chi.loc[col1, col2] = chi2_val

#Spearman correlation
spearman_corr = hotel[numerical_columns].corr(method='spearman')

#Kendall correlation
kendall_corr = hotel[numerical_columns].corr(method='kendall')

#Cramer's V
cramer_v = pd.DataFrame(data=[(0 for _ in range(len(categorical_columns))) for _ in rang
                        columns=list(categorical_columns))
cramer_v.set_index(pd.Index(list(categorical_columns)), inplace = True)

for col1 in categorical_columns:
    for col2 in categorical_columns:
        if col1 != col2:
            contingency_table = pd.crosstab(hotel[col1], hotel[col2])
            cramer = stats.contingency.association(contingency_table, method="cramer")
            cramer_v.loc[col1, col2] = cramer
            cramer_v.loc[col2, col1] = cramer

titles= ['Chi2 values for categorical variables','P values for categorical variables', '
         'kendall correlation for numerical variables', 'Cramer\'s V for categorical var
values = [resultant_chi, resultant, spearman_corr, kendall_corr, cramer_v]

figure = plt.figure()
figure.set_size_inches(5, 5)
figure, axes = plt.subplots(nrows=3, ncols=2, figsize=(14, 14))
figure.subplots_adjust(hspace = 1, wspace = 0.4)
sns.set(font_scale=.7)

for i, ax in enumerate(axes.flat, start=1):
    ax.set_title(titles[i-1])
    sns.heatmap(values[i-1], annot=True, cmap='coolwarm', ax=ax, square=True, cbar_kws={
    if i == len(titles):
        break

plt.tight_layout()
plt.show()
```

```
<Figure size 500x500 with 0 Axes>
```



Skewness can be calculated using various methods, whereas the most commonly used method is Pearson's coefficient. We usee three more methods:

- Spearman's correlation : To measure nonlinear correlation
- Kendall's correlation : Is a non-parametric measure of relationships between columns
- Cramers's correlation

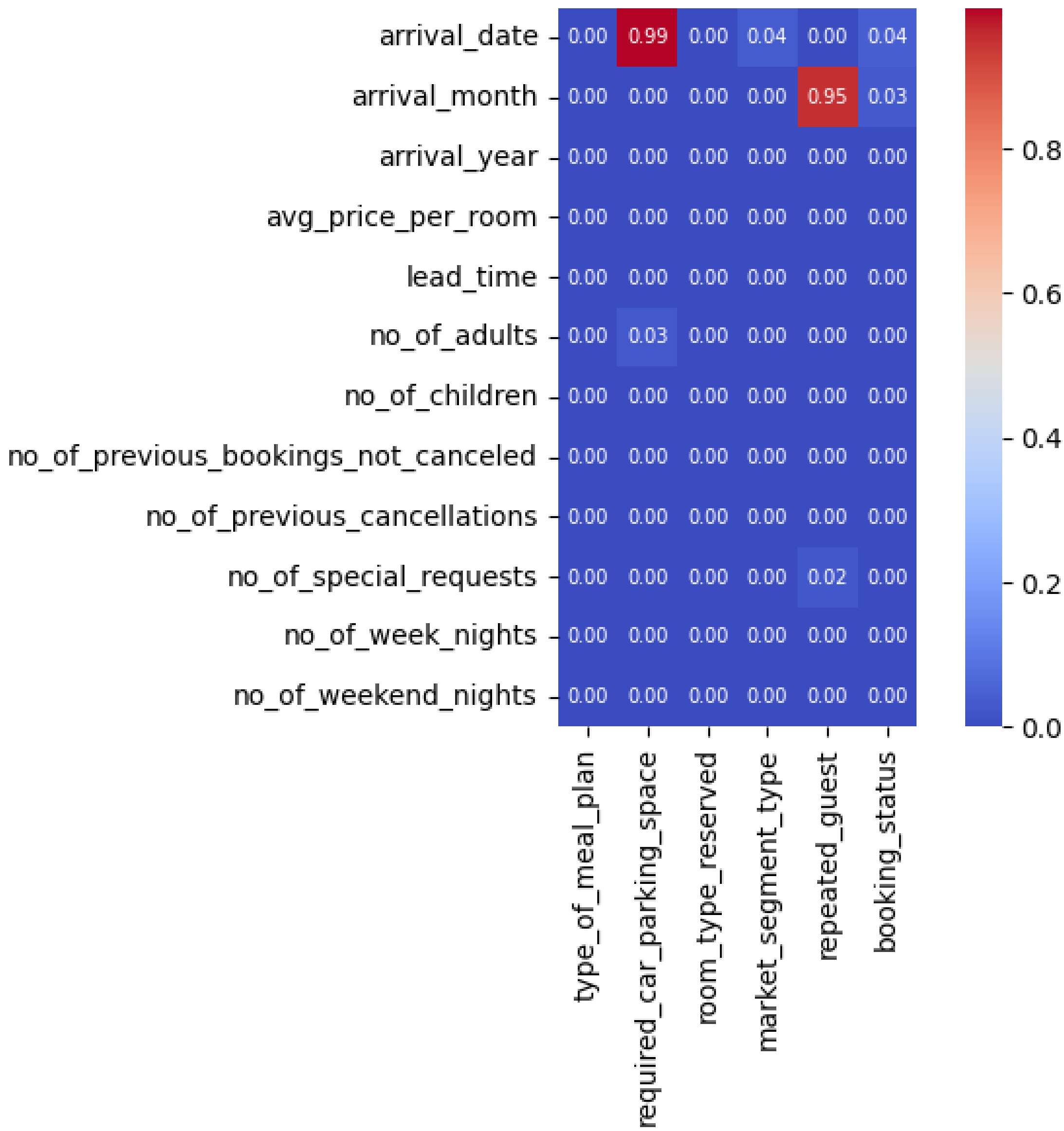
Tests that compare multiple populations

Anova test

- Is a statistical test used to determine whether there are significant differences between the means of two or more groups

In [6]:
```
# anova heatmap for numerical variables and categorical variables
anova_df = pd.DataFrame(data=[(0 for _ in range(len(categorical_columns))) for _ in rang
                        columns=list(categorical_columns))
anova_df.set_index(pd.Index(list(numerical_columns)), inplace = True)

for col1 in numerical_columns:
    for col2 in categorical_columns:
        groups = hotel.groupby(col2)[col1].apply(list).values
        pvalue = stats.f_oneway(*groups)[1]
        anova_df.loc[col1, col2] = pvalue

sns.heatmap(anova_df, annot=True, cmap='coolwarm', square=True, annot_kws={'size': 7}, f
```

Out[6]:
```
<Axes: >
```



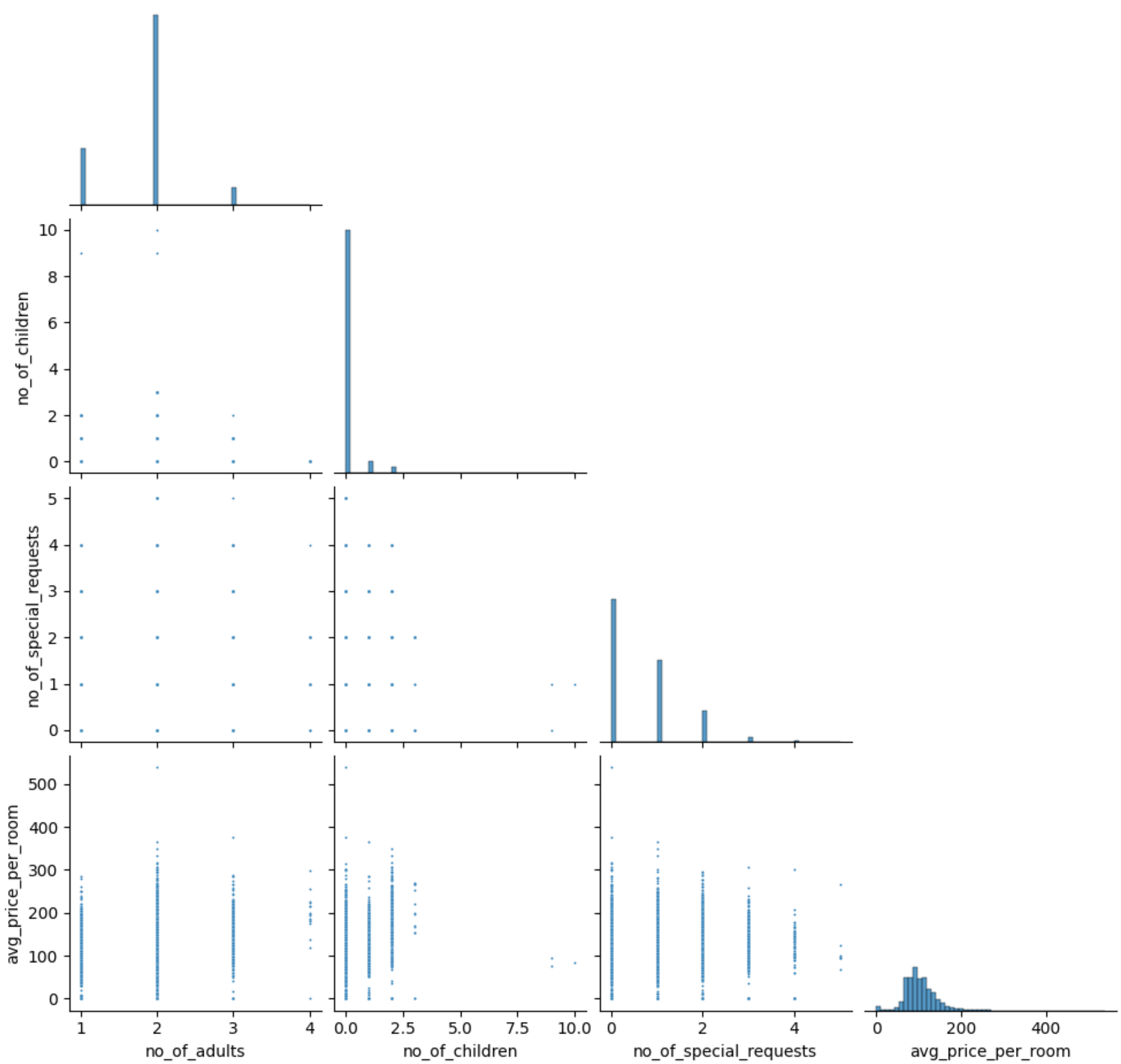
- A big p-value means a weaker association: as we can observe between {'required_car_parking_space'} and {'arrival_date'} and also between {'repeated_quests'} and {'arrival_month'}.

Scatterplots

In [81]:
```
scatter_columns = [
    "no_of_adults",
    "no_of_children",
```

```
    "no_of_special_requests",
    "avg_price_per_room",
]

sns.pairplot(hotel[scatter_columns], diag_kind='hist', corner=True, plot_kws={'s': 2}, d
```

Out[81]: <seaborn.axisgrid.PairGrid at 0x1d5b3ad69e0>

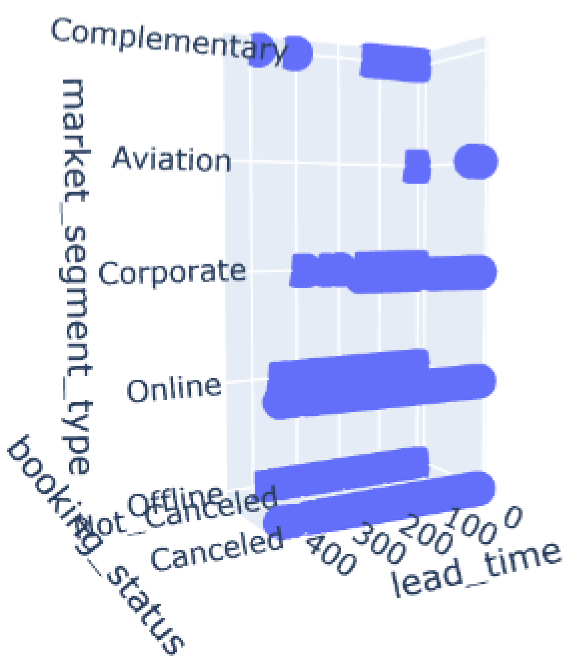


3D Scatterplots

In [56]:

```
%matplotlib widget

# 3d grafic (atribute numerice)
def scatter_3d_plot(name1, name2, name3):
    fig = px.scatter_3d(hotel, x=name1, y=name2, z=name3)
    fig.show()

first_column = hotel["lead_time"].copy()
second_column = hotel["booking_status"].copy()
third_column = hotel["market_segment_type"].copy()
```

```
first_name = "lead_time"
second_name = "booking_status"
third_name = "market_segment_type"

scatter_3d_plot(first_name, second_name, third_name)
```

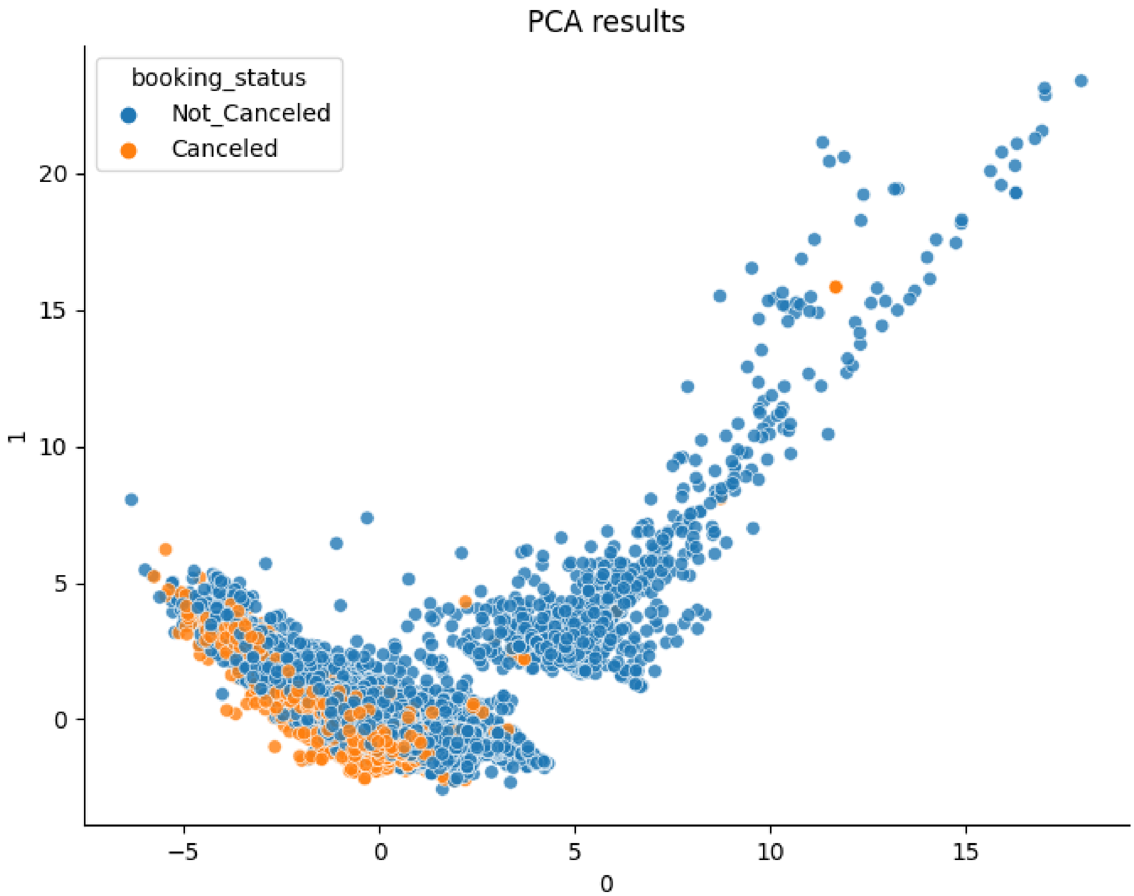


- We can observe that when lead_time is big there are more chances to cancel the rezervation and also for the Complementary segment there are no cancellations.

Principal component analysis

In [70]:
```
import numpy as np
x= hotel.drop("booking_status", axis = 1)
y = hotel["booking_status"]

numcols = x.select_dtypes(["int64","float64"]).columns
catcols = x.select_dtypes("object").columns

from sklearn.preprocessing import StandardScaler

# standardize the data: Standardize features by removing the mean and scaling to unit va
scaler = StandardScaler()

for catcol in catcols:
    x[catcol] = x[catcol].astype("category").cat.codes

# apply the scaler to the data
scaler.fit(x)
# transform the data
x_scaled = scaler.transform(x)

train_features = x_scaled

from sklearn.decomposition import PCA
# use all variables (except the target variable and the ID variable)
PCA = PCA(n_components = 17)

PCA.fit(train_features)

x_pca =PCA.transform(train_features)
pca = pd.DataFrame(x_pca)
pca_y = pca.join(y)

plt.figure(figsize=(8,6))
sns.scatterplot(data = pca_y, x=0, y=1, hue="booking_status", alpha = .8)
sns.despine(top = True, right = True, left = False, bottom = False)
plt.title("PCA results")

plt.show()

plt.figure(figsize=(10, 10))
plt.plot(PCA.explained_variance_ratio_, 'o-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.xticks(range(17))

# cumulative variance explained plot
plt.figure(figsize=(10, 10))
plt.plot(np.cumsum(PCA.explained_variance_ratio_), 'o-', linewidth=2)
plt.title('Cumulative Variance Explained Plot')
plt.xlabel('Principal Component')
plt.ylabel('Cumulative Proportion of Variance Explained')
plt.xticks(range(17))
```

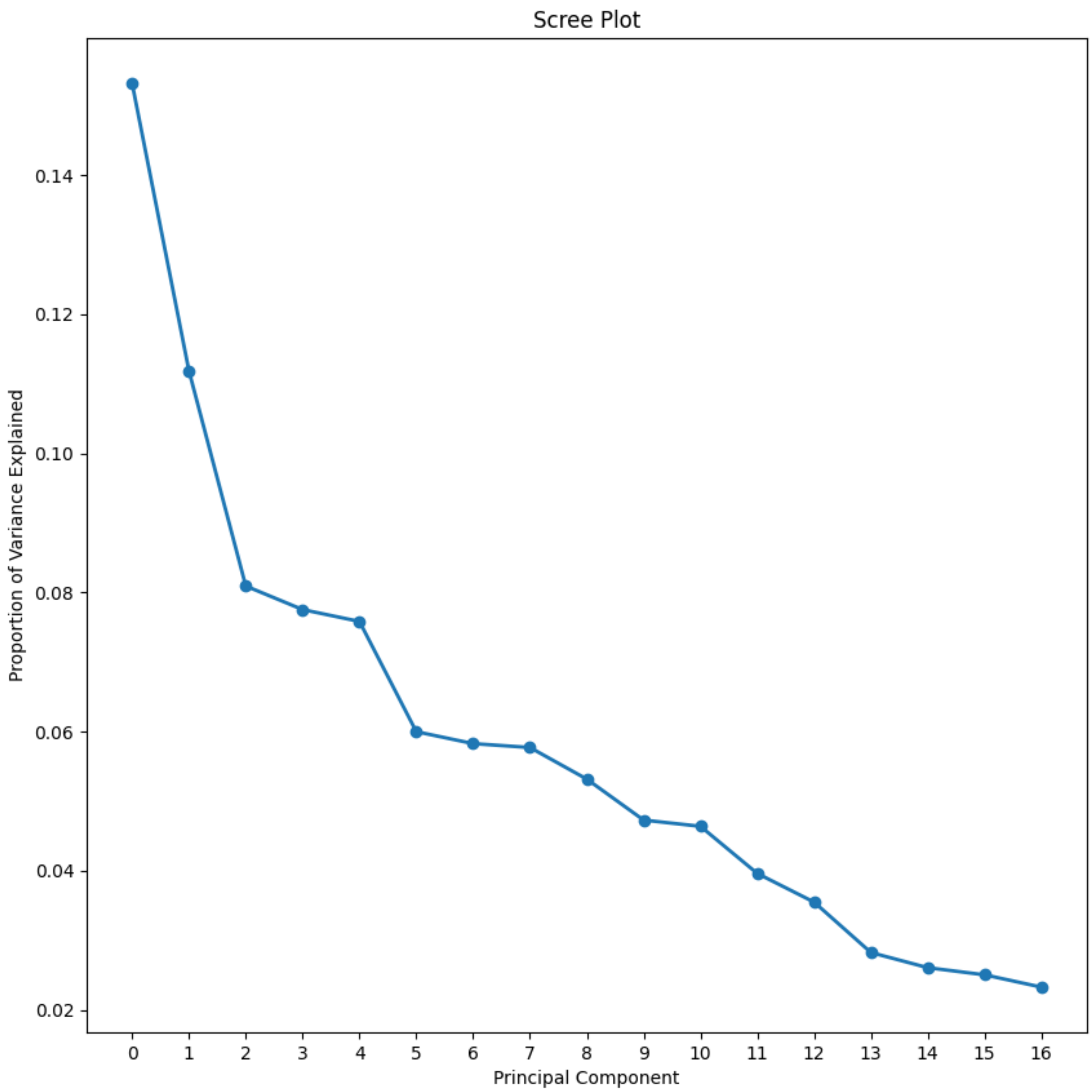
Figure



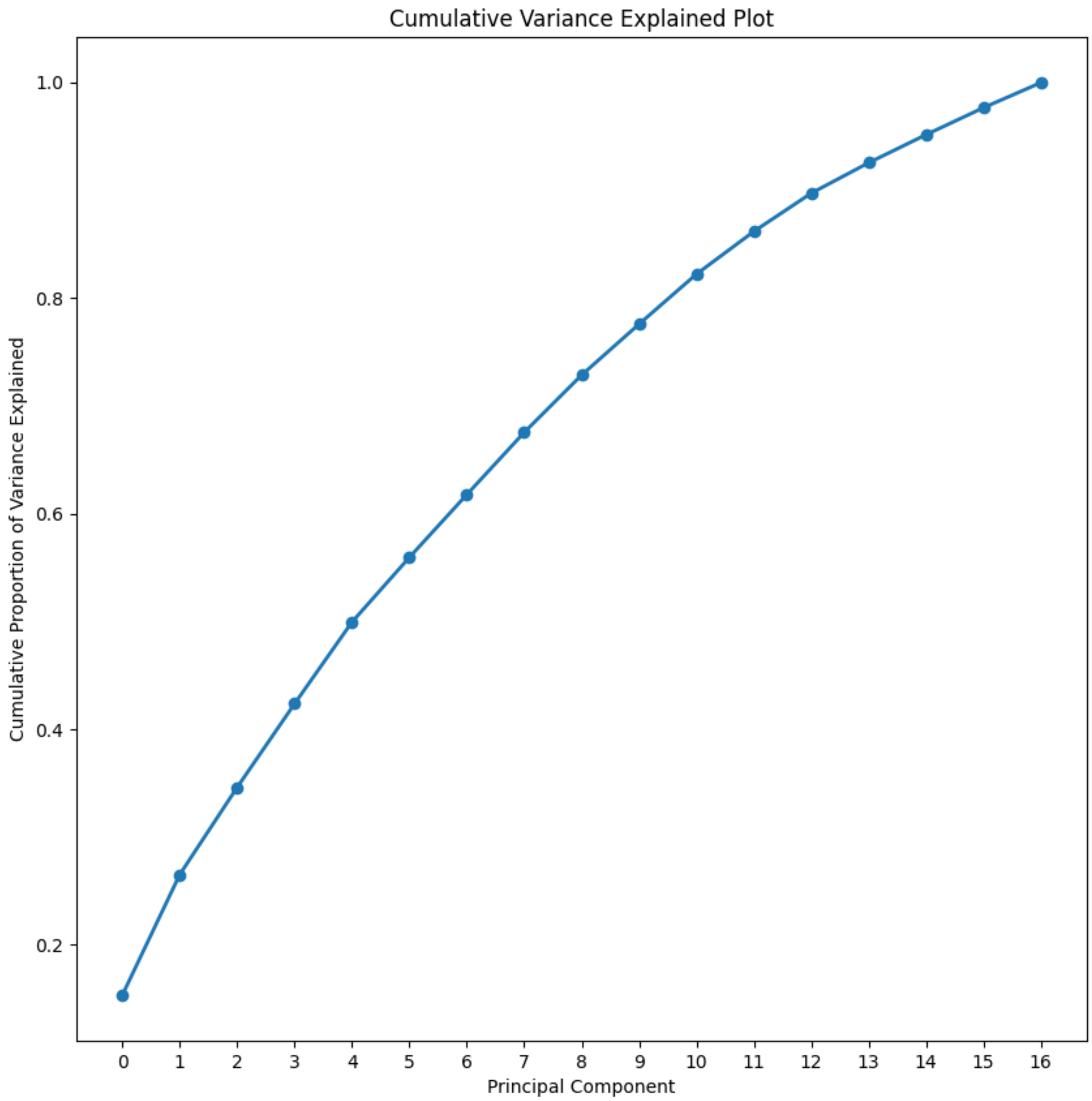

```
Out[70]: ([<matplotlib.axis.XTick at 0x21ef21e3d60>,
  <matplotlib.axis.XTick at 0x21ef21e3d30>,
  <matplotlib.axis.XTick at 0x21ef21e3a30>,
  <matplotlib.axis.XTick at 0x21ef223c430>,
  <matplotlib.axis.XTick at 0x21ef223cee0>,
  <matplotlib.axis.XTick at 0x21ef223d990>,
  <matplotlib.axis.XTick at 0x21ef223dae0>,
  <matplotlib.axis.XTick at 0x21ef223e020>,
  <matplotlib.axis.XTick at 0x21ef223e980>,
  <matplotlib.axis.XTick at 0x21ef223f160>,
  <matplotlib.axis.XTick at 0x21ef223f940>,
  <matplotlib.axis.XTick at 0x21ef223f220>,
  <matplotlib.axis.XTick at 0x21ef223fbe0>,
  <matplotlib.axis.XTick at 0x21ef223d420>,
  <matplotlib.axis.XTick at 0x21ef2258b80>,
  <matplotlib.axis.XTick at 0x21ef2259360>,
  <matplotlib.axis.XTick at 0x21ef2259b40>],
 [Text(0, 0, '0'),
  Text(1, 0, '1'),
  Text(2, 0, '2'),
  Text(3, 0, '3'),
  Text(4, 0, '4'),
  Text(5, 0, '5'),
  Text(6, 0, '6'),
  Text(7, 0, '7'),
  Text(8, 0, '8'),
  Text(9, 0, '9'),
  Text(10, 0, '10'),
  Text(11, 0, '11'),
  Text(12, 0, '12'),
  Text(13, 0, '13'),
  Text(14, 0, '14'),
```

```
Text(15, 0, '15'),
Text(16, 0, '16')])
```

Figure



Figure



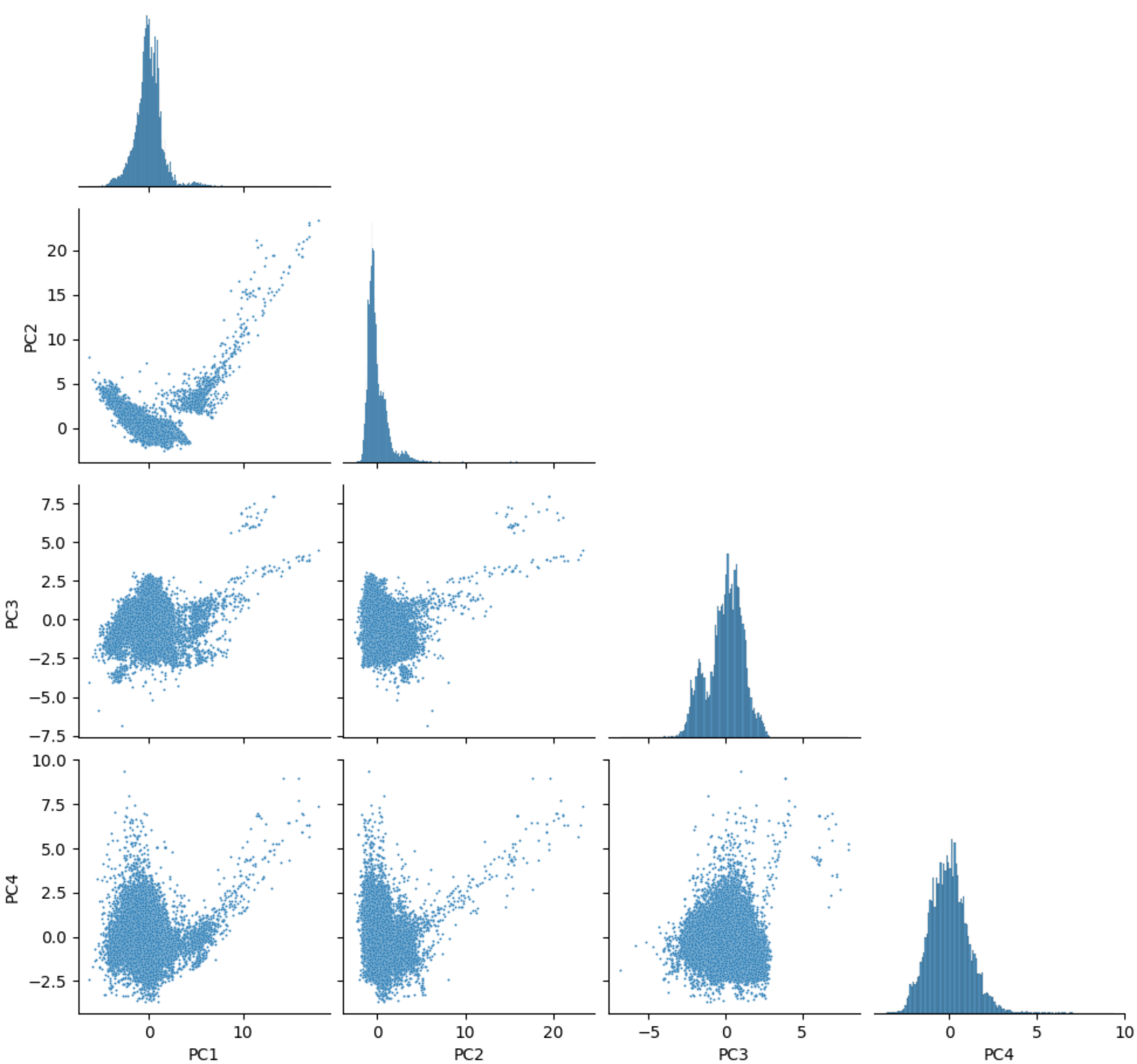
In [63]:

```
#pairplot for the first 8 principal components
hotel_pca_df = pd.DataFrame(x_pca, columns=['PC'+str(i) for i in range(1, 18)])
sns.pairplot(hotel_pca_df.iloc[:, :4], diag_kind='hist', corner=True, plot_kws={'s': 2})
```

Out[63]:

```
<seaborn.axisgrid.PairGrid at 0x21ee638be50>
```

Figure



In [71]:
```
pd.DataFrame(PCA.components_.T, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'
                                          'PC13', 'PC14', 'PC15', 'PC16', 'PC17'], index=
```

Out[71]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
no_of_adults	-0.335822	0.062516	0.080924	0.143069	0.242173	-0.165031	-0.181787
no_of_children	-0.227627	0.292916	-0.205202	-0.073128	-0.269268	0.055828	0.085381
no_of_weekend_nights	-0.123851	-0.007627	0.110123	0.402729	0.127010	0.538634	0.264046
no_of_week_nights	-0.142931	-0.039292	-0.022431	0.546675	0.076324	0.162703	0.192391
type_of_meal_plan	-0.008477	-0.154152	0.362948	-0.330885	0.409266	-0.021466	0.015510
required_car_parking_space	0.006822	0.185091	-0.003715	-0.192042	0.059443	-0.050232	-0.160401
room_type_reserved	-0.316836	0.379941	-0.189727	0.059778	-0.232815	-0.005634	0.007475
lead_time	-0.043844	-0.218205	0.047633	0.508440	0.003004	-0.473692	-0.341688
arrival_year	-0.142674	0.120346	0.603250	0.110401	-0.240700	-0.229954	-0.033305

arrival_month	-0.013581	-0.066349	-0.561829	0.084901	0.437130	-0.174426	-0.158393
arrival_date	-0.034338	0.024149	0.079599	0.005379	-0.072860	0.524236	-0.820592
market_segment_type	-0.416326	0.015369	0.210966	-0.100473	0.313348	0.010393	0.073495
repeated_guest	0.365210	0.405125	0.008740	0.061577	0.079739	-0.007723	-0.015795
no_of_previous_cancellations	0.234091	0.394430	0.146783	0.161715	0.218797	-0.099126	-0.009429
no_of_previous_bookings_not_canceled	0.306480	0.441705	0.098688	0.151901	0.174464	-0.044643	-0.032601
avg_price_per_room	-0.412665	0.276261	-0.081892	-0.108554	-0.086501	-0.196033	-0.059179
no_of_special_requests	-0.223090	0.218792	-0.015182	-0.101824	0.420692	0.130791	0.016740

- We can observe that for the PC1 the most important variables are the "repeated_guest", "no_of_previous_bookings_not_canceled", "no_of_previous_cancellations".

Non-linear bidimensional mappings

- t-SNE: One of the most widely used techniques for visualization, but its performance suffers with large datasets

In [25]:
```
dataset = hotel[numerical_columns]
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=250)
tsne_results = tsne.fit_transform(dataset)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 36275 samples in 0.183s...
[t-SNE] Computed neighbors for 36275 samples in 2.277s...
[t-SNE] Computed conditional probabilities for sample 1000 / 36275
[t-SNE] Computed conditional probabilities for sample 2000 / 36275
[t-SNE] Computed conditional probabilities for sample 3000 / 36275
[t-SNE] Computed conditional probabilities for sample 4000 / 36275
[t-SNE] Computed conditional probabilities for sample 5000 / 36275
[t-SNE] Computed conditional probabilities for sample 6000 / 36275
[t-SNE] Computed conditional probabilities for sample 7000 / 36275
[t-SNE] Computed conditional probabilities for sample 8000 / 36275
[t-SNE] Computed conditional probabilities for sample 9000 / 36275
[t-SNE] Computed conditional probabilities for sample 10000 / 36275
[t-SNE] Computed conditional probabilities for sample 11000 / 36275
[t-SNE] Computed conditional probabilities for sample 12000 / 36275
[t-SNE] Computed conditional probabilities for sample 13000 / 36275
[t-SNE] Computed conditional probabilities for sample 14000 / 36275
[t-SNE] Computed conditional probabilities for sample 15000 / 36275
[t-SNE] Computed conditional probabilities for sample 16000 / 36275
[t-SNE] Computed conditional probabilities for sample 17000 / 36275
[t-SNE] Computed conditional probabilities for sample 18000 / 36275
[t-SNE] Computed conditional probabilities for sample 19000 / 36275
[t-SNE] Computed conditional probabilities for sample 20000 / 36275
[t-SNE] Computed conditional probabilities for sample 21000 / 36275
[t-SNE] Computed conditional probabilities for sample 22000 / 36275
[t-SNE] Computed conditional probabilities for sample 23000 / 36275
[t-SNE] Computed conditional probabilities for sample 24000 / 36275
[t-SNE] Computed conditional probabilities for sample 25000 / 36275
[t-SNE] Computed conditional probabilities for sample 26000 / 36275
[t-SNE] Computed conditional probabilities for sample 27000 / 36275
[t-SNE] Computed conditional probabilities for sample 28000 / 36275
[t-SNE] Computed conditional probabilities for sample 29000 / 36275
[t-SNE] Computed conditional probabilities for sample 30000 / 36275
[t-SNE] Computed conditional probabilities for sample 31000 / 36275
[t-SNE] Computed conditional probabilities for sample 32000 / 36275
```

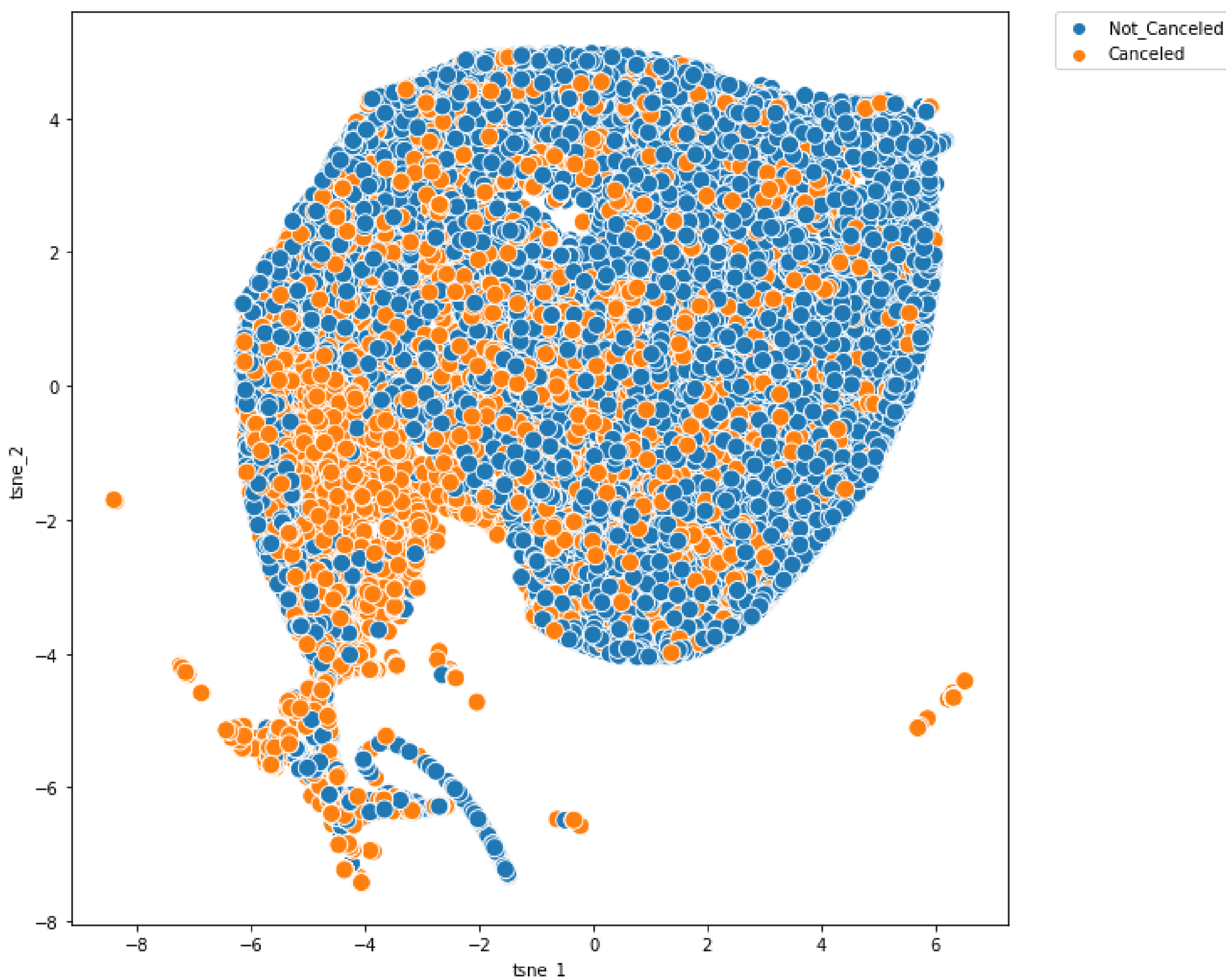
```
[t-SNE] Computed conditional probabilities for sample 33000 / 36275
[t-SNE] Computed conditional probabilities for sample 34000 / 36275
[t-SNE] Computed conditional probabilities for sample 35000 / 36275
[t-SNE] Computed conditional probabilities for sample 36000 / 36275
[t-SNE] Computed conditional probabilities for sample 36275 / 36275
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration: 79.940964
[t-SNE] KL divergence after 251 iterations: 17976931348623157081452742373170435679807056
7525844996598917476803157260780028538760589558632766878171540458953514382464234321326889
4641827684675467035375169860499105765512820762454900903893289440758685084551339423045832
3690322294816580855933212334827479782620414472316873817718091929988125040402618412485836
8.000000
```

In [51]:

```python
# plot the result of TSNE with the label color coded
tsne_result_df = pd.DataFrame({'tsne_1': tsne_results[:,0], 'tsne_2': tsne_results[:,1],
fig, ax = plt.subplots(1)
fig.set_size_inches(10, 10)
sns.scatterplot(x='tsne_1', y='tsne_2', hue='label', data=tsne_result_df, ax=ax,s=120)
# lim = (tsne_results.min()-5, tsne_results.max()+5)
# ax.set_xlim(lim)
# ax.set_ylim(lim)
# ax.set_aspect('equal')
ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)
```

Out[51]:

```
<matplotlib.legend.Legend at 0x15b97141ac0>
```



- From the above results it is difficult to analyze the formed clusters (also the results are strongly influenced by the hyer-parameters used), and more visual results is obtained using uMap.

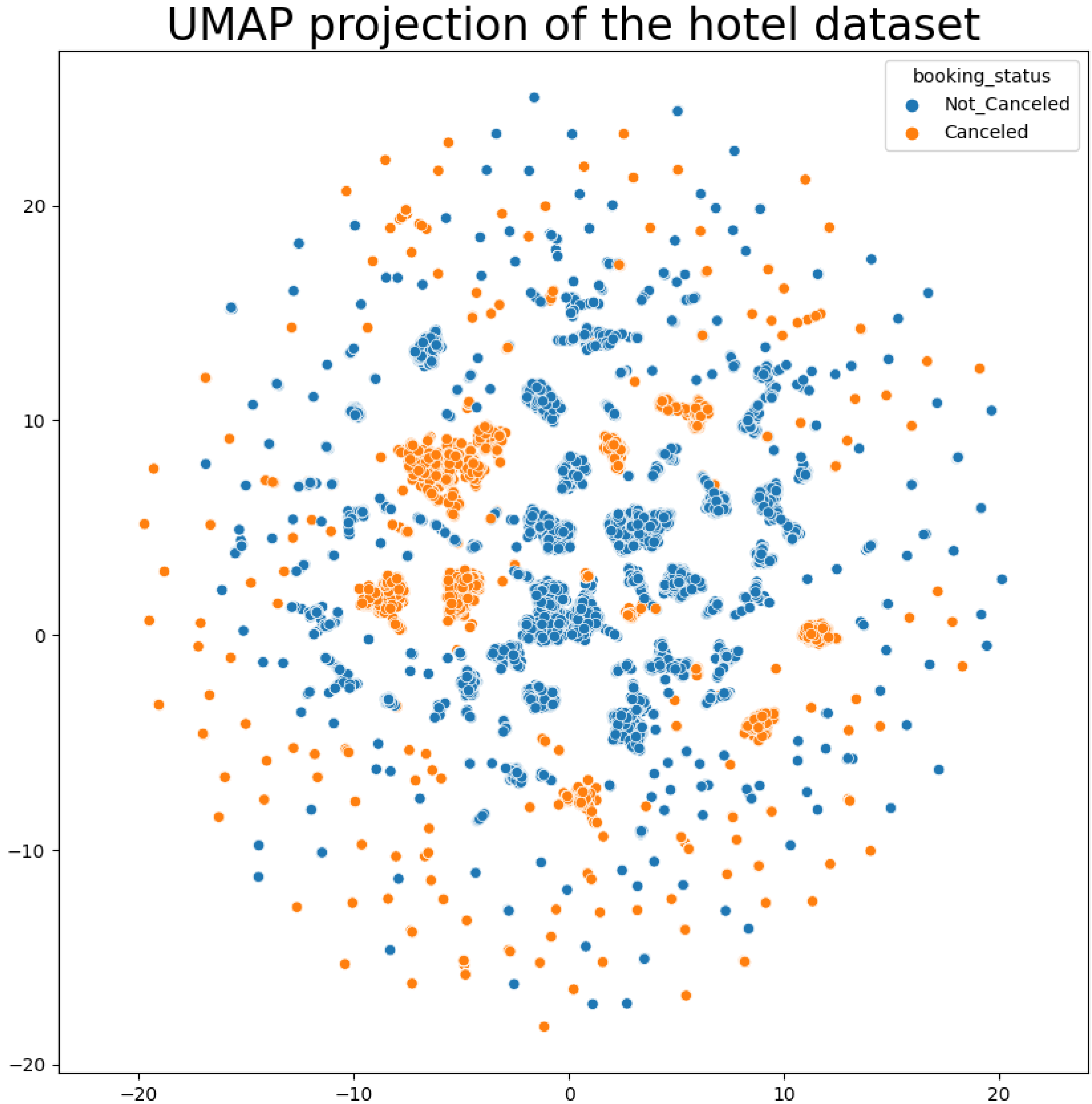
uMap

In [24]:
```
from sklearn.preprocessing import StandardScaler
data = hotel

for cat_col in categorical_columns:
    data[cat_col] = data[cat_col].astype('category').cat.codes

scaled_data = StandardScaler().fit_transform(data)

reducer = umap.UMAP()

embedding = reducer.fit_transform(scaled_data)
```

```
d:\ProgramData\Anaconda3\envs\DataMining\lib\site-packages\sklearn\manifold\_spectral_em
bedding.py:274: UserWarning: Graph is not fully connected, spectral embedding may not wo
rk as expected.
  warnings.warn(
```

In [37]:
```
plt.figure(figsize=(10, 10))
sns.scatterplot(x=embedding[:, 0], y=embedding[:, 1], hue=hotel.booking_status)
plt.gca().set_aspect('equal', 'datalim')
plt.title('UMAP projection of the hotel dataset', fontsize=24)
```

Out[37]:
```
Text(0.5, 1.0, 'UMAP projection of the hotel dataset')
```



T-test

In [77]:
```
temp_df = hotel.copy()

for cat_col in categorical_columns:
    temp_df[cat_col] = temp_df[cat_col].astype('category').cat.codes

t_stat, p_value = stats.ttest_ind(temp_df[hotel.booking_status == 'Canceled'], temp_df[h
for i in range(len(p_value)):
    print(hotel.columns[i], p_value[i], t_stat[i])

print(hotel[hotel.booking_status == 'Canceled']['lead_time'].median())
print(hotel[hotel.booking_status == 'Not_Canceled']['lead_time'].median())
print("---------------------")
print(hotel[hotel.booking_status == 'Canceled']['no_of_special_requests'].median())
print(hotel[hotel.booking_status == 'Not_Canceled']['no_of_special_requests'].median())
```

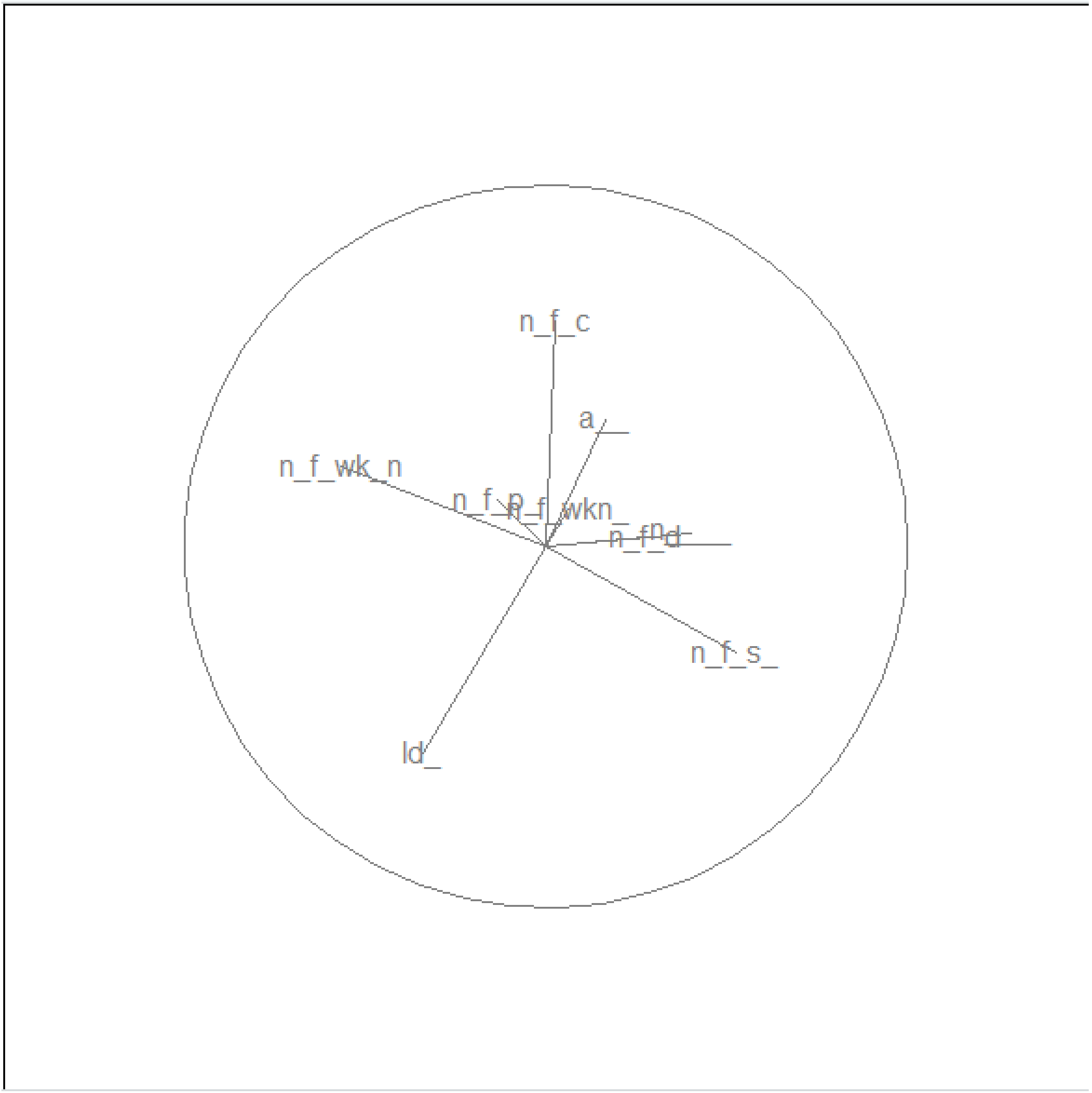
```
no_of_adults 1.9345253475709744e-68 17.535449990894485
no_of_children 2.0034501853140845e-10 6.364306623228386
```

```
no_of_weekend_nights 1.0093281704434595e-29 11.339848841821798
no_of_week_nights 3.157038245417444e-61 16.565709776138743
type_of_meal_plan 3.788817921395253e-07 5.0807289629243675
required_car_parking_space 4.683163282445571e-92 -20.40816019130208
room_type_reserved 1.3591073688645307e-05 4.351326442396521
lead_time 0.0 80.43905526483107
arrival_year 0.0 39.979169563518624
arrival_month 0.018947559599713164 -2.346696720839616
arrival_date 0.040037683236619305 2.053474703644861
market_segment_type 6.403309326293967e-182 28.957512801871914
repeated_guest 9.261449100647606e-178 -28.63541972000149
no_of_previous_cancellations 2.3066357827498886e-14 -7.635491077530978
no_of_previous_bookings_not_canceled 3.852262564191205e-60 -16.402036037094916
avg_price_per_room 1.0983691249622683e-174 28.396205201360804
no_of_special_requests 0.0 -56.196940514819836
booking_status 0.0 -inf
122.0
39.0
---------------------
0.0
1.0
---------------------
4
4
```

```
C:\Users\cezar\AppData\Local\Temp\ipykernel_6800\1998838173.py:6: RuntimeWarning: Precis
ion loss occurred in moment calculation due to catastrophic cancellation. This occurs wh
en the data are nearly identical. Results may be unreliable.
  t_stat, p_value = stats.ttest_ind(temp_df[hotel.booking_status == 'Canceled'], temp_df
[hotel.booking_status == 'Not_Canceled'], equal_var=False)
```

Projection Pursuit

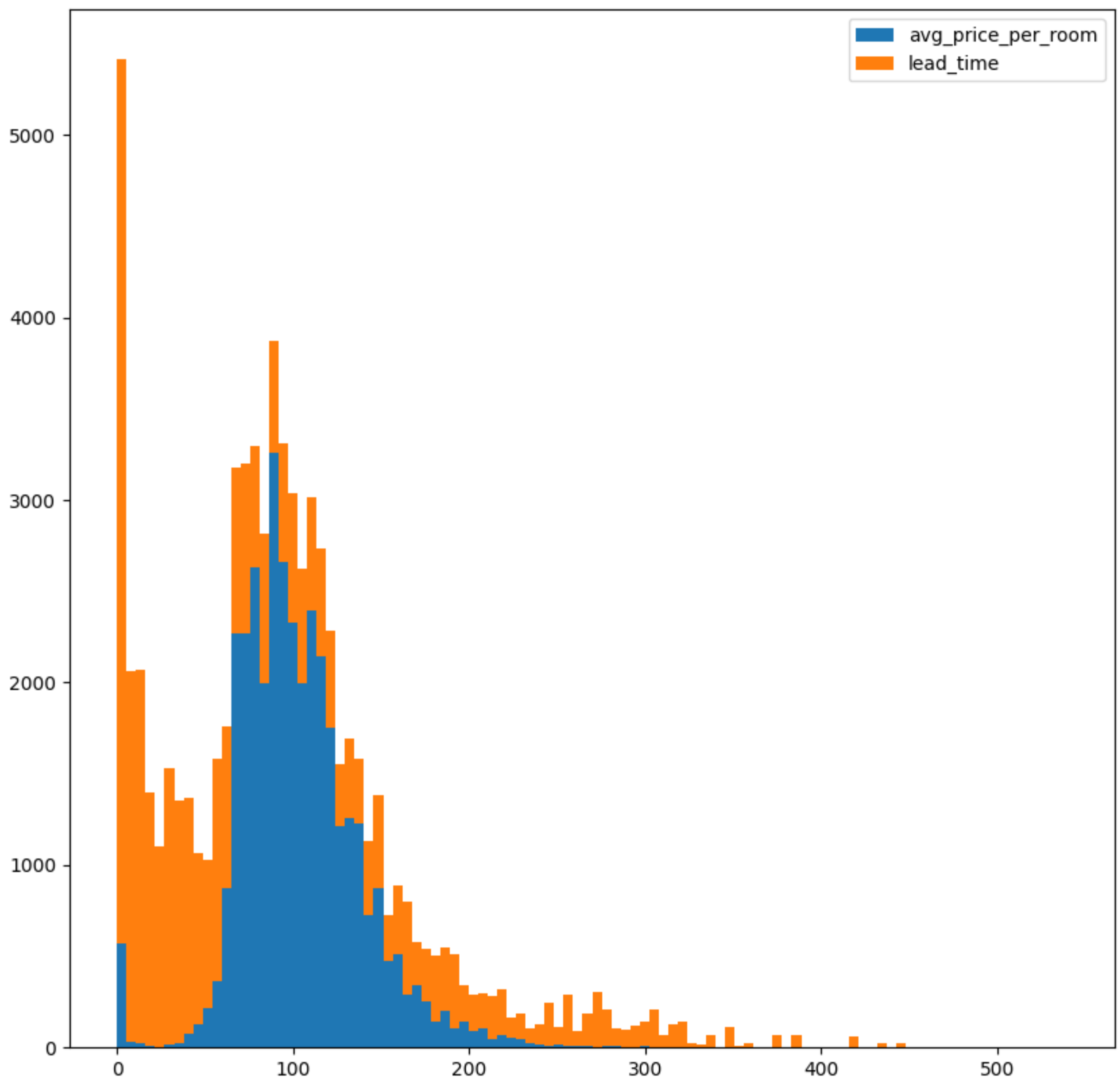
library(tourr) data = read.csv("D:\\DM\\Hotel Reservations.csv") data = na.omit(data) data = data[, !names(data) %in% c("Booking_ID","type_of_meal_plan", "required_car_parking_space", "room_type_reserved", "market_segment_type", "repeated_guest", "booking_status", "arrival_year", "arrival_month", "arrival_date")] animate_xy(data, col=data$booking_status) # the tour is grand_tour by default



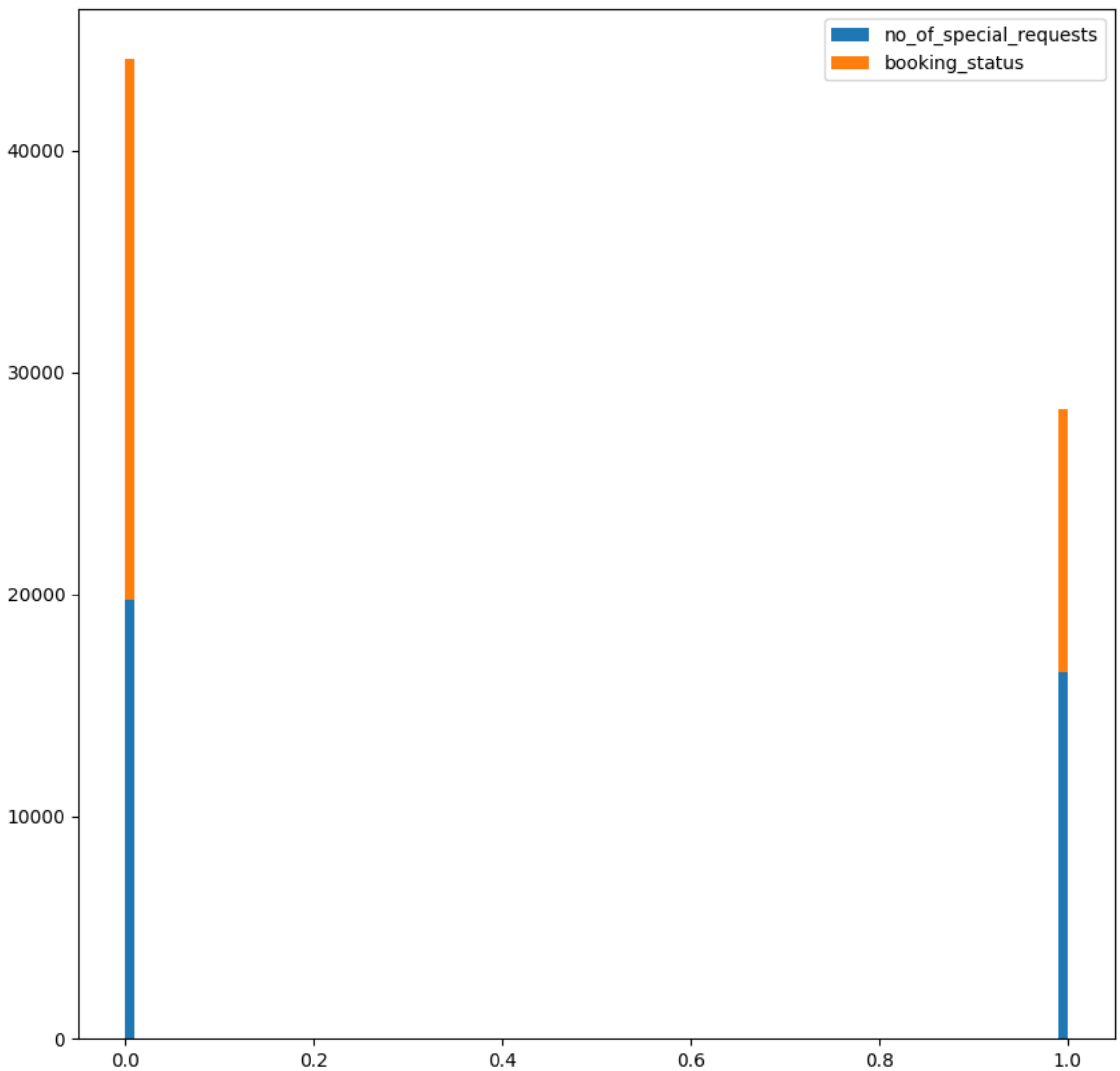
Stacked histograms

In [77]:
```
plt.figure(figsize=(10, 10))
plt.hist([hotel["avg_price_per_room"], hotel["lead_time"]], bins = 100, stacked=True, la
plt.legend()
plt.show()

plt.figure(figsize=(10, 10))
binary_request = pd.Series(np.where(hotel["no_of_special_requests"] > 0, 1, 0))
binary_status = pd.Series(np.where(hotel["booking_status"] == 'Canceled', 1, 0))
plt.hist([binary_request, binary_status], bins = 100, stacked=True, label = [hotel["no_o
plt.legend()
plt.show()
```

Figure


avg_price_per_room
lead_time
0
1000
2000
3000
4000
5000
0
100
200
300
400
500

Figure

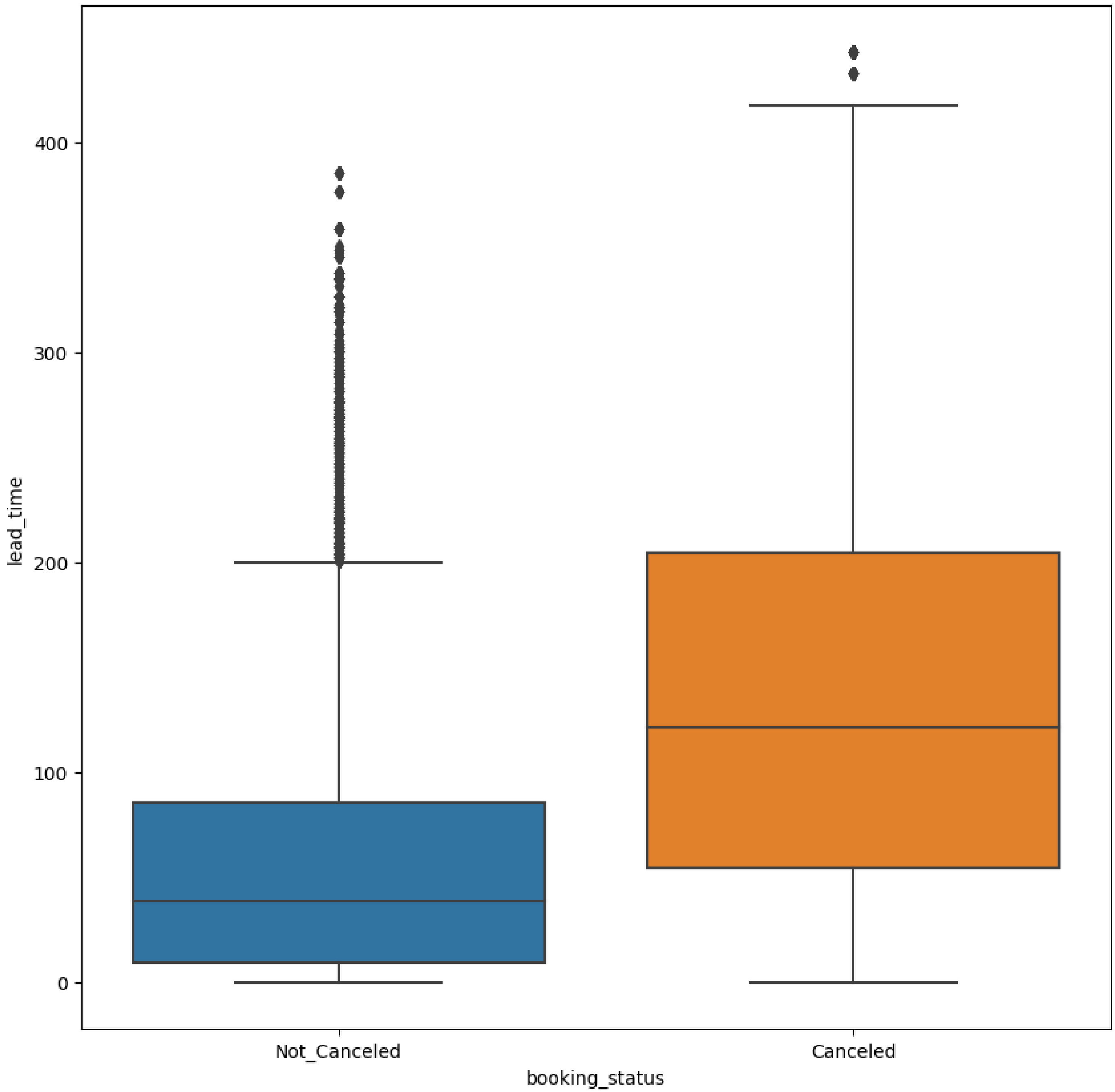


Conditional boxplots

In [71]:

```
#conditioned boxplot
plt.figure(figsize=(10, 10))
sns.boxplot(x="booking_status", y="lead_time", data=hotel)
plt.show()

#determine the number of outliers
Q1 = hotel[hotel.booking_status == 'Not_Canceled']["lead_time"].quantile(0.25)
Q3 = hotel[hotel.booking_status == 'Not_Canceled']["lead_time"].quantile(0.75)
IQR = Q3 - Q1
print(IQR)

count = hotel[hotel.booking_status == 'Not_Canceled']["lead_time"][hotel.lead_time > Q3
        hotel[hotel.booking_status == 'Not_Canceled']["lead_time"][hotel.lead_time < Q1
print("Outliers : ", count)

#determine the number of outliers
# for entry in hotel[hotel.booking_status == 0]["lead_time"]:
```

```
#     if entry > Q3 + 1.5*IQR or entry < Q1 - 1.5*IQR:


Q1 = hotel[hotel.booking_status == 'Canceled']['lead_time'].quantile(0.25)
Q3 = hotel[hotel.booking_status == 'Canceled']['lead_time'].quantile(0.75)
IQR = Q3 - Q1
print(IQR)

count = hotel[hotel.booking_status == 'Canceled']["lead_time"][hotel.lead_time > Q3 + 1.
        hotel[hotel.booking_status == 'Canceled']["lead_time"][hotel.lead_time < Q1 - 1.

print("Outliers : ", count)
```




```
76.0
Outliers :  1035
150.0
Outliers :  42
```

Market segment type

In [152…

```
pd.crosstab(hotel.market_segment_type, hotel.booking_status)
```

Out[152]:

booking_status	Canceled	Not_Canceled
market_segment_type		
Aviation	37	88
Complementary	0	391
Corporate	220	1797
Offline	3153	7375
Online	8475	14739

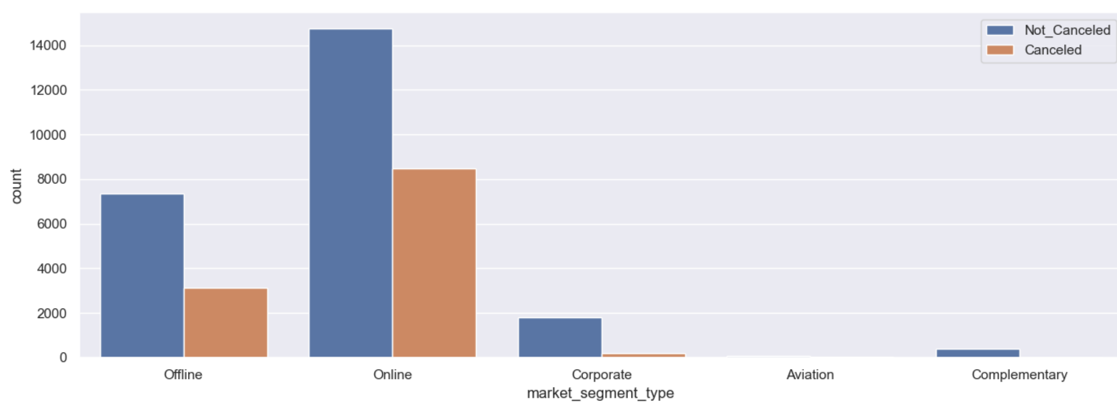
In [154…

```
plt.figure(figsize=[15,5])
sns.set()
sns.countplot(x = 'market_segment_type', hue = 'booking_status', data = hotel)
plt.legend(loc = 1)
```

Out[154]:

```
<matplotlib.legend.Legend at 0x1d581543be0>
```

Figure



We can see how the reservations are distributed according to the type of package:

- reservations made online have the highest cancellation rate: 37.5, probably because cancellation is much easier
- "corporate" and "complementary" reservations have the lowest cancellation rate: 10.9% and 0% respectively
- aviation reservations have a cancellation rate of 29%, which may be due to unforeseen events

In [156…

```
hotel.groupby('arrival_month')['booking_status'].value_counts(normalize = True)
```

Out[156]:

```
arrival_month  booking_status
1              Not_Canceled      0.976331
               Canceled          0.023669
2              Not_Canceled      0.747653
               Canceled          0.252347
3              Not_Canceled      0.703138
               Canceled          0.296862
4              Not_Canceled      0.636330
               Canceled          0.363670
5              Not_Canceled      0.635104
               Canceled          0.364896
6              Not_Canceled      0.596940
               Canceled          0.403060
7              Not_Canceled      0.550000
               Canceled          0.450000
8              Not_Canceled      0.609756
```

```
               Canceled          0.390244
9              Not_Canceled      0.666450
               Canceled          0.333550
10             Not_Canceled      0.646417
               Canceled          0.353583
11             Not_Canceled      0.706376
               Canceled          0.293624
12             Not_Canceled      0.866931
               Canceled          0.133069
Name: booking_status, dtype: float64
```

- We notice that the months with the highest cancellation rate are June, July and August, with July having the highest cancellation rate: 45%

In [ ]:

```
#conditioned boxplot
plt.figure(figsize=(10, 10))
sns.boxplot(x="booking_status", y="avg_price_per_room", data=hotel)
plt.show()

#determine the number of outliers
Q0 = hotel[hotel.booking_status == 'Not_Canceled']["avg_price_per_room"].quantile(0.0)
Q1 = hotel[hotel.booking_status == 'Not_Canceled']["avg_price_per_room"].quantile(0.25)
Q3 = hotel[hotel.booking_status == 'Not_Canceled']["avg_price_per_room"].quantile(0.75)
Q4 = hotel[hotel.booking_status == 'Not_Canceled']["avg_price_per_room"].quantile(1.0)
IQR = Q3 - Q1
print("Q0", Q0)
print("Q4", Q4)
print(IQR)

count = hotel[hotel.booking_status == 'Not_Canceled']["avg_price_per_room"][hotel.avg_pr
        hotel[hotel.booking_status == 'Not_Canceled']["avg_price_per_room"][hotel.avg_pr
print("Outliers : ", count)

#determine the number of outliers
# for entry in hotel[hotel.booking_status == 0]["avg_price_per_room"]:
#     if entry > Q3 + 1.5*IQR or entry < Q1 - 1.5*IQR:

Q0 = hotel[hotel.booking_status == 'Canceled']["avg_price_per_room"].quantile(0.0)
Q1 = hotel[hotel.booking_status == 'Canceled']['avg_price_per_room'].quantile(0.25)
Q3 = hotel[hotel.booking_status == 'Canceled']['avg_price_per_room'].quantile(0.75)
Q4 = hotel[hotel.booking_status == 'Canceled']["avg_price_per_room"].quantile(1.0)
IQR = Q3 - Q1
print("Q0", Q0)
print("Q4", Q4)
print(IQR)

count = hotel[hotel.booking_status == 'Canceled']["avg_price_per_room"][hotel.avg_price_
        hotel[hotel.booking_status == 'Canceled']["avg_price_per_room"][hotel.avg_price_

print("Outliers : ", count)
```