Introduction to Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a technique in data science that is used to understand the structure of the data, detecting the outliers and anomalies, discovering patterns, and testing hypotheses. By employing a variety of statistical tools and graphical techniques, it allows data scientists and students to gain insights that guide subsequent analysis and modeling efforts.

Key Techniques

1) **Descriptive Statistics:**
   EDA allows data analysts to gain the statistical measures of the data like calculating mean, median, mode, standard deviation, and percentiles which helps to summarize the central tendency and variability of the data. Moreover, it helps in understanding the distribution of categorical variables through counts and proportions.
2) **Data Visualization:**
   EDA helps to visualize the data (graphical interpretation) using histograms, box plots, scatter plots, bar charts, correlation matrices, and so on.
3) **Handling Missing Values:**
   It allows analysts to identify missing data points (values) and decide upon how to handle missing data, whether through deletion, mean/median/mode imputation, or other sophisticated methods.
4) **Outlier Detection:**
   EDA helps in the identification of anomalies or outliers (values that are deviated from the normal) by making the use of statistical techniques and visualizations like box plots and histograms.
5) **Feature Engineering:**
   It eases the process of data transformation, normalizing the data, and creation of new features based on existing ones to capture more information.

Tools Used

1) **Numpy:**

   It is an open-source library that facilitates efficient numerical operation on large quantities of data. It provides support for large multi-dimensional arrays and matrices, and is the foundation of **pandas** library. Numpy arrays are more flexible than normal python lists and are used extensively in data manipulation and analysis.

2) **Pandas:**

   It is a powerful library for data manipulation and analysis. It provides data structures and functions to efficiently handle structured data, including tabular data such as spreadsheets and SQL tables. It is built on top of **NumPy** and provides data structures such as **Series** (a one-dimensional labeled array) and **DataFrame** (a two-dimensional labeled data structure with columns of potentially different types). Pandas is particularly useful for data cleaning, filtering, grouping, and merging.

3) **Matplotlib:**

   Matplotlib is a Python visualization library that provides a comprehensive set of tools for creating high-quality 2D and 3D plots, charts, and graphs. It is often used in conjunction with **Pandas** to visualize data. Matplotlib provides a wide range of visualization options, including line plots, scatter plots, bar charts, histograms, and more.

4) **Scikit-learn:**

   It is used for machine learning that provides a wide range of algorithms for **classification**, **regression**, **clustering**, and other tasks. It is built on top of **NumPy** and **SciPy**, and is particularly useful for building predictive models and data mining. Scikit-learn provides tools for data preprocessing, feature selection, and model evaluation, as well as a wide range of algorithms for specific tasks such as classification, regression, clustering, and more.

Questions

1. Data Inspection and Cleaning

a) Check for missing values and handle them appropriately (e.g., imputation, dropping rows/columns).

b) Check for duplicates and remove them if necessary.

c) Identify and handle outliers, if any.

d) Check for inconsistencies in the data (e.g., invalid values, typos).

First import the required libraries and load the data.

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import warnings
     import seaborn as sns

     warnings.filterwarnings('ignore')
```

```
[2]: # Data Ingestion
     csv_path = 'hotel_bookings.csv'
     df_hotel = pd.read_csv(csv_path)
```

Data Inspection and Cleaning

```
[3]: df_hotel.shape
```

```
[3]: (119390, 32)
```

```
[4]: df_hotel.columns
```

```
[4]: Index(['hotel', 'is_canceled', 'lead_time', 'arrival_date_year',
           'arrival_date_month', 'arrival_date_week_number',
           'arrival_date_day_of_month', 'stays_in_weekend_nights',
           'stays_in_week_nights', 'adults', 'children', 'babies', 'meal',
           'country', 'market_segment', 'distribution_channel',
           'is_repeated_guest', 'previous_cancellations',
           'previous_bookings_not_canceled', 'reserved_room_type',
           'assigned_room_type', 'booking_changes', 'deposit_type', 'agent',
           'company', 'days_in_waiting_list', 'customer_type', 'adr',
           'required_car_parking_spaces', 'total_of_special_requests',
           'reservation_status', 'reservation_status_date'],
          dtype='object')
```

```
[6]: # Check if missing values exist
     df_hotel.isna().sum()
```

```
[6]: hotel                               0
     is_canceled                         0
     lead_time                           0
     arrival_date_year                   0
     arrival_date_month                  0
     arrival_date_week_number            0
     arrival_date_day_of_month           0
     stays_in_weekend_nights             0
     stays_in_week_nights                0
     adults                              1
     children                            4
     babies                              0
     meal                                0
     country                           507
     market_segment                      2
     distribution_channel                1
     is_repeated_guest                   0
     previous_cancellations              0
```

Data Imputation

Rows/Cols with minimum missing values (like in the range of 1-10) can be deleted because it does not provide much significance and simplies the data analysis process.

[8]:
```
cols_to_delete = ['adults', 'children', 'market_segment', 'distribution_channel', 'reserved_room_type', 'assigned_room_type', 'reser
df_hotel.dropna(subset=cols_to_delete, inplace=True)
```

Note: Filled the missing values (country-column) with the most repeated country

[9]:
```
country_mode = df_hotel.country.mode()[0]
df_hotel['country'].fillna(country_mode, inplace=True)
```

[10]:
```
df_hotel[df_hotel['country'].isna()] # null values check
```

[10]:

hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stay

0 rows × 32 columns

Note: Filled the missing values (deposit_type-column) with the most repeated frequency

[11]:
```
df_hotel['deposit_type'].describe()
```

[11]:
```
count         119358
unique             3
top       No Deposit
freq          104620
Name: deposit_type, dtype: object
```

[12]:
```
mode_dt = df_hotel['deposit_type'].mode()[0]
df_hotel['deposit_type'].fillna(mode_dt, inplace=True)
```

Note: Filled the agent and company column with 0 because they are company ids and agents ids, thus 0 means undefined company id and agent id.

[13]:
```
df_hotel['agent'].fillna(0, inplace=True)
df_hotel['company'].fillna(0, inplace=True)
```

Note: Filled the customer type with the most repeated frequency (mode)

[14]:
```
mode_ct = df_hotel['customer_type'].mode()[0]
df_hotel['customer_type'].fillna(mode_ct, inplace=True)
```

[15]:
```
# Check if null values still exist?
df_hotel.isna().sum()
```

[15]:
```
hotel                             0
is_canceled                       0
lead_time                         0
arrival_date_year                 0
arrival_date_month                0
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          0
```

Duplicate Data Handling

[16]:
```
df_hotel.duplicated().sum() # 32009 rows are duplicates
```

[16]:
```
32009
```

[17]:
```
# Copy the dataset in a separate variable and drop duplicated values
data = df_hotel.copy()
data.drop_duplicates(inplace=True)
```

[18]:
```
f"Duplicated rows: {data.duplicated().sum()}" # after deleting
```

[18]:
```
'Duplicated rows: 0'
```

Finding out outliers using Inter-quartile range (IQR)

[20]:
```
def find_outliers(df, col):
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)

    IQR = q3 - q1
    lower_bound = q1 - 1.5 * IQR
    upper_bound = q3 + 1.5 * IQR

    condition = (df[col] < lower_bound) | (df[col] > upper_bound)
    outliers = df[condition]

    return outliers
```

[21]:
```
lead_time_outliers = find_outliers(data_clean, 'lead_time')
lead_time_outliers
```

2. Descriptive Statistics

a) Calculate summary statistics (mean, median, mode, standard deviation, etc.) for numerical columns like lead_time, stays_in_weekend_nights, stays_in_week_nights, adults, children, babies, previous_cancellations, previous_bookings_not_canceled, days_in_waiting_list, adr, required_car_parking_spaces, and total_of_special_requests.

b) Display value counts and frequencies for categorical columns like hotel, country, market_segment, distribution_channel, is_repeated_guest, reserved_room_type, assigned_room_type, deposit_type, agent, company, customer_type, reservation_status.

Descriptive Statistics

[24]:
```
data_clean.describe()
```

[24]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children
count	87370.000000	87370.000000	87370.000000	87370.000000	87370.000000	87370.000000	87370.000000
mean	79.919995	26.840254	15.815474	1.005288	2.625489	1.875884	0.138652
min	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	11.000000	16.000000	8.000000	0.000000	1.000000	2.000000	0.000000
50%	49.000000	27.000000	16.000000	1.000000	2.000000	2.000000	0.000000
75%	125.000000	37.000000	23.000000	2.000000	4.000000	2.000000	0.000000
max	737.000000	53.000000	31.000000	19.000000	50.000000	55.000000	10.000000
std	86.071919	13.674301	8.834487	1.031931	2.053477	0.626496	0.455915

[25]:
```
data_clean.describe(include='category')
```

[25]:

	hotel	is_canceled	arrival_date_year	arrival_date_month	meal	country	market_segment	distribution_channel	is_repeated_guest	reserved_roon
count	87370	87370	87370	87370	87370	87370	87370	87370	87370	
unique	2	2	3	12	5	177	7	5	2	
top	City Hotel	0	2016	August	BB	PRT	Online TA	TA/TO	0	
freq	53428	63348	42384	11251	67956	27880	51613	69133	83955	

3. Data Visualization

a) Create histograms or box plots for numerical columns to visualize the distribution and identify potential outliers.

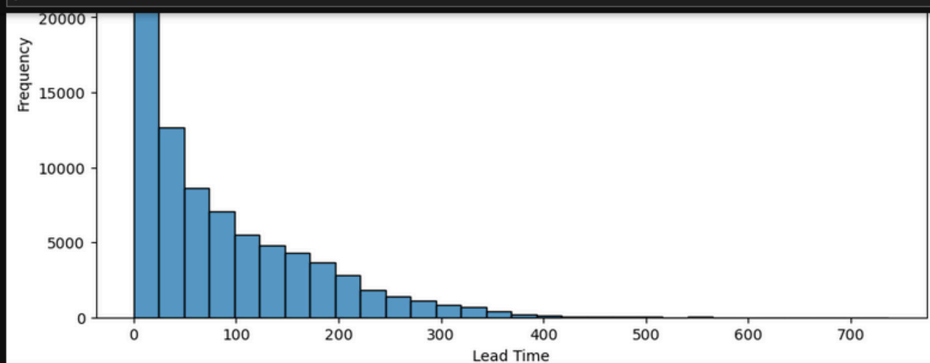
b) Use bar plots or pie charts to visualize the distribution of categorical columns.

c) Create scatter plots or heatmaps to explore relationships between numerical columns.

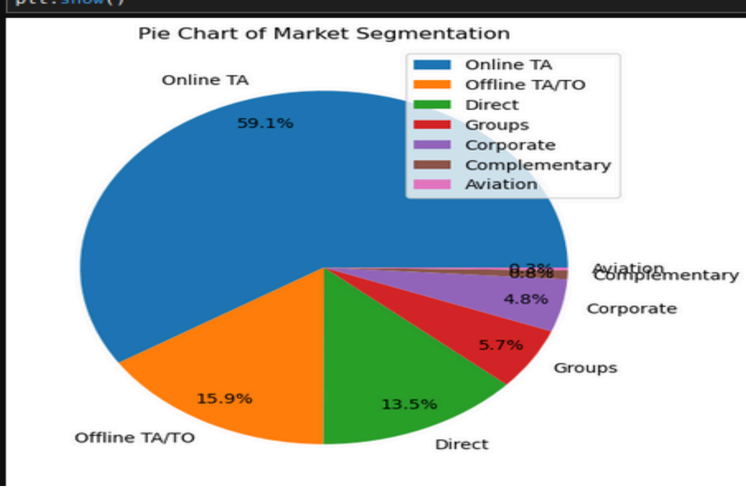
d) Use line plots to visualize trends over time for columns like arrival_date_year, arrival_date_month, arrival_date_week_number, and arrival_date_day_of_month.

Data Visualization

[26]:
```
# Histogram of lead_time
plt.figure(figsize=(10, 6))
sns.histplot(data_clean['lead_time'], bins=30)
plt.title('Histogram of Lead Time')
plt.xlabel('Lead Time')
plt.ylabel('Frequency')
plt.show()
```

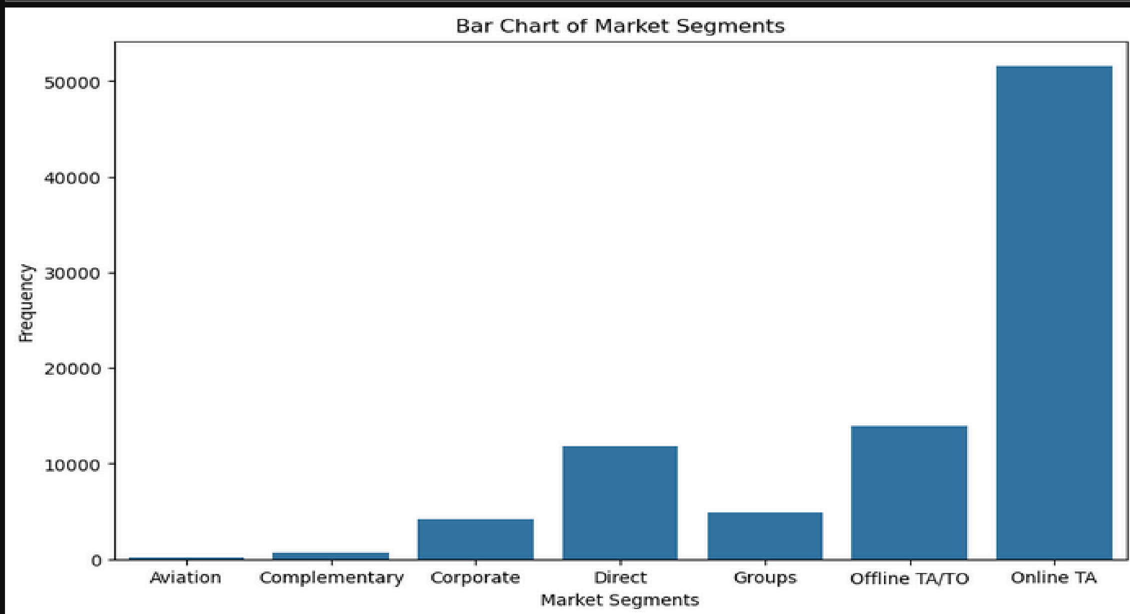


[30]:
```
# Pie chart for market segment
plt.figure(figsize=(6, 6))
segments = list(data_clean['market_segment'].value_counts())
plt.pie(segments,
        labels=data_clean['market_segment'].value_counts().index,
        autopct='%.1f%%',
        pctdistance=0.85,  # Adjust percentage label distance
       )
plt.title('Pie Chart of Market Segmentation')
plt.legend()
plt.show()
```



[31]:

```
# Bar plot for Market Segments
plt.figure(figsize=(10, 6))
sns.countplot(x=data_clean['market_segment'])
plt.title('Bar Chart of Market Segments')
plt.xlabel('Market Segments')
plt.ylabel('Frequency')
plt.show()
```



4. Correlation Analysis

a) Calculate the correlation matrix to identify potential relationships between numerical columns. (use pandas df.corr()).

b) Visualize the correlation matrix using a heatmap.

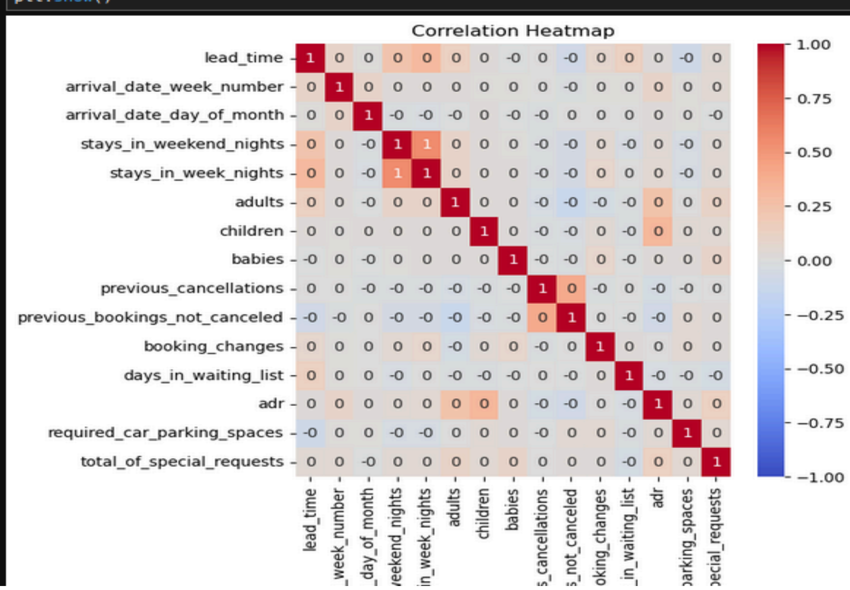
Correlation Analysis

[35]:
```python
correlation = data_clean.corr(method='pearson', numeric_only=True)
correlation
```

[35]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults
lead_time	1.000000	0.101215	0.009890	0.234904	0.309983	0.140401
arrival_date_week_number	0.101215	1.000000	0.093569	0.026773	0.027692	0.024150
arrival_date_day_of_month	0.009890	0.093569	1.000000	-0.017907	-0.028359	-0.001108
stays_in_weekend_nights	0.234904	0.026773	-0.017907	1.000000	0.555470	0.088190
stays_in_week_nights	0.309983	0.027692	-0.028359	0.555470	1.000000	0.095470
adults	0.140401	0.024150	-0.001108	0.088190	0.095470	1.000000
children	0.028560	0.013391	0.015812	0.028560	0.030476	0.023664
babies	-0.003645	0.014249	-0.000393	0.013667	0.016008	0.016629
previous_cancellations	0.005347	0.007188	-0.008540	-0.020641	-0.018789	-0.042116
previous_bookings_not_canceled	-0.078961	-0.020838	0.000153	-0.056663	-0.058520	-0.120948
booking_changes	0.076908	0.011872	0.006291	0.050301	0.085023	-0.048108
days_in_waiting_list	0.132924	0.014240	0.006794	-0.031846	0.001751	-0.015530
adr	0.023337	0.098055	0.022412	0.038879	0.053187	0.248923
required_car_parking_spaces	-0.086634	0.008888	0.009199	-0.042936	-0.044317	0.007754
total_of_special_requests	0.034133	0.046460	-0.001639	0.032493	0.037855	0.112639

[36]:
```python
plt.figure(figsize=(6, 6))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.0f', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```



5. **Categorical Data Analysis**

a) Perform one-hot encoding or label encoding for categorical columns if required for further analysis.

b) Analyze categorical columns like hotel, country, market_segment, distribution_channel, reserved_room_type, assigned_room_type, deposit_type, agent, company, customer_type, and reservation_status to identify patterns or trends.

c) Create contingency tables or mosaic plots to analyze the relationship between categorical columns.

Categorical Data Analysis

[37]:
```
data_clean.describe(include='category')
```

[37]:

	hotel	is_canceled	arrival_date_year	arrival_date_month	meal	country	market_segment	distribution_channel	is_repeated_guest	reserved_roon
count	87370	87370	87370	87370	87370	87370	87370	87370	87370	
unique	2	2	3	12	5	177	7	5	2	
top	City Hotel	0	2016	August	BB	PRT	Online TA	TA/TO	0	
freq	53428	63348	42384	11251	67956	27880	51613	69133	83955	

[38]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
```

[53]:
```
label_encoder = LabelEncoder()
one_hot_encoder = OneHotEncoder()
```

[52]:
```
data_clean['hotel'].value_counts()
```

[52]:
```
hotel
City Hotel      53428
Resort Hotel    33942
Name: count, dtype: int64
```

There are two types of hotel. So, we can encode them as: City Hotel = 0 and Resort Hotel = 1

[105]:
```
hotel_enc = one_hot_encoder.fit_transform(data_clean[['hotel']]).toarray()
enc_df = pd.DataFrame(hotel_enc, columns=one_hot_encoder.get_feature_names_out(['hotel']))

# Concatenate the encoded dataset with the origin one
joined_df = pd.concat([enc_df, data_clean.reset_index(drop=True)], axis=1)
```

[136]:
```
# These are the columns that are to be encoded
cols_to_encode = ['meal', 'market_segment', 'distribution_channel', 'deposit_type', 'customer_type', 'reservation_status']

for col in cols_to_encode:
    joined_df[col] = label_encoder.fit_transform(joined_df[col])
```

[191]:
```
# Verify whether the columns are encoded properly or not!
joined_df[cols_to_encode].head()
```

[191]:

	meal	market_segment	distribution_channel	deposit_type	customer_type	reservation_status
0	0	3	1	0	2	1
1	0	3	1	0	2	1
2	0	3	1	0	2	1
3	0	2	0	0	2	1
4	0	6	3	0	2	1

Analyzing Categorical Columns

[145]:
```
categorical_cols = data_clean.describe(include='category').columns
```

[147]:
```
# Summary Statistics of Categorical columns
data_clean.describe(include='category')
```

[147]:

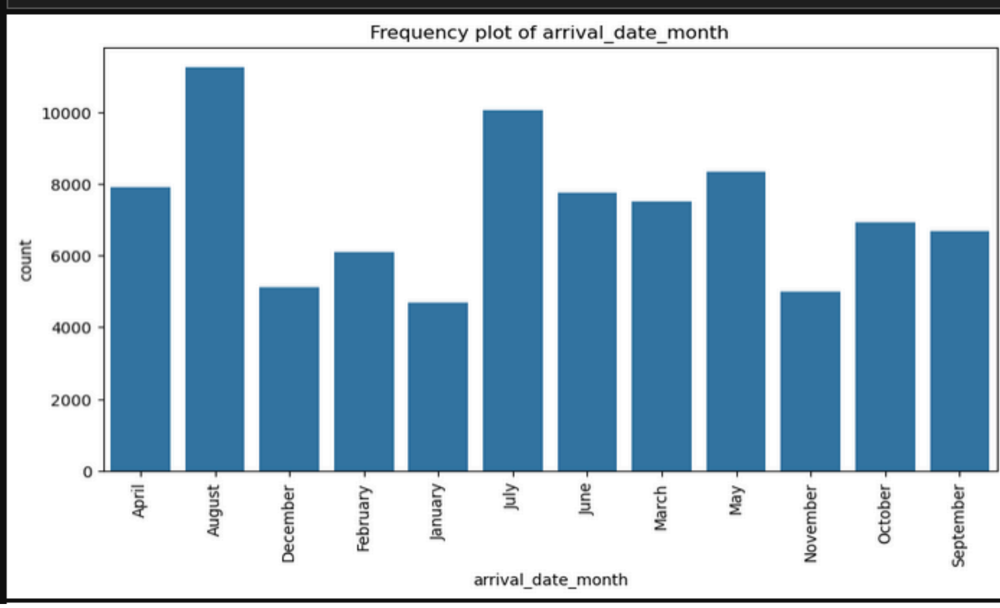
	hotel	is_canceled	arrival_date_year	arrival_date_month	meal	country	market_segment	distribution_channel	is_repeated_guest	reserved_
count	87370	87370	87370	87370	87370	87370	87370	87370	87370	
unique	2	2	3	12	5	177	7	5	2	
top	City Hotel	0	2016	August	BB	PRT	Online TA	TA/TO	0	
freq	53428	63348	42384	11251	67956	27880	51613	69133	83955	

[152]:
```
for col in categorical_cols:
    print(f'Frequency of items in {col.title()}:')
    print(data_clean[col].value_counts())
    print()
```

```
Frequency of items in Hotel:
hotel
City Hotel      53428
Resort Hotel    33942
Name: count, dtype: int64

Frequency of items in Is_Canceled:
is_canceled
0    63348
1    24022
Name: count, dtype: int64
```

[162]:
```
# Bar plots for categorical columns
for col in categorical_cols:
    plt.figure(figsize=(10, 5))
    sns.countplot(data=data_clean, x=col)
    plt.title(f'Frequency plot of {col}')
    plt.xticks(rotation=90)
    plt.show()
```



[163]:
```python
# Contingency table between hotel and market_segment
contingency_table = pd.crosstab(data_clean['hotel'], data_clean['market_segment'])
contingency_table
```

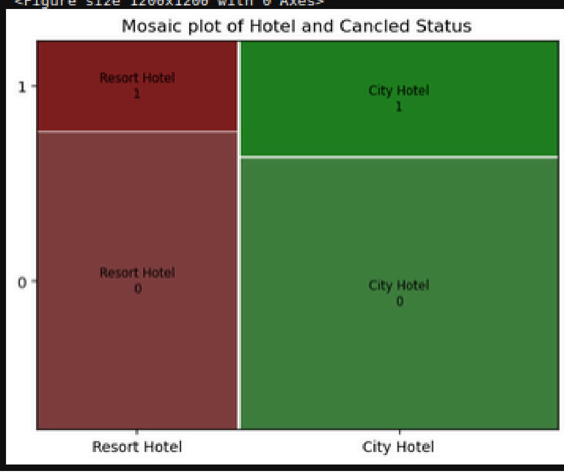
[163…

market_segment	Aviation	Complementary	Corporate	Direct	Groups	Offline TA/TO	Online TA
hotel							
City Hotel	227	513	2227	5558	2638	7272	34993
Resort Hotel	0	189	1979	6239	2305	6610	16620

[166]:
```python
from statsmodels.graphics.mosaicplot import mosaic
```

[229]:
```python
# Create the mosiac plot
plt.figure(figsize=(12, 12))
mosaic(data_clean, ['hotel', 'is_canceled'])
plt.title('Mosaic plot of Hotel and Cancled Status')
plt.show()
```

<Figure size 1200x1200 with 0 Axes>



6. Time Series Analysis

a) Analyze the arrival_date_year, arrival_date_month, arrival_date_week_number, and arrival_date_day_of_month columns to identify seasonality or trends in bookings over time.

b) Create time series plots or decompose the time series to understand the trend, seasonality, and residuals.

Time Series Analysis

[193]:
```python
# Copy the dataset in a new variable
new_df = data_clean.copy()
```

[260]:
```python
# Combine the separate arrival year, month, and day as arrival date
new_df['arrival_date'] = pd.to_datetime(new_df['arrival_date_year'].astype(str) + '-' + new_df['arrival_date_month'].astype(str)
```

[261]:
```python
new_df[['arrival_date', 'arrival_date_year', 'arrival_date_month', 'arrival_date_day_of_month']].head()
```

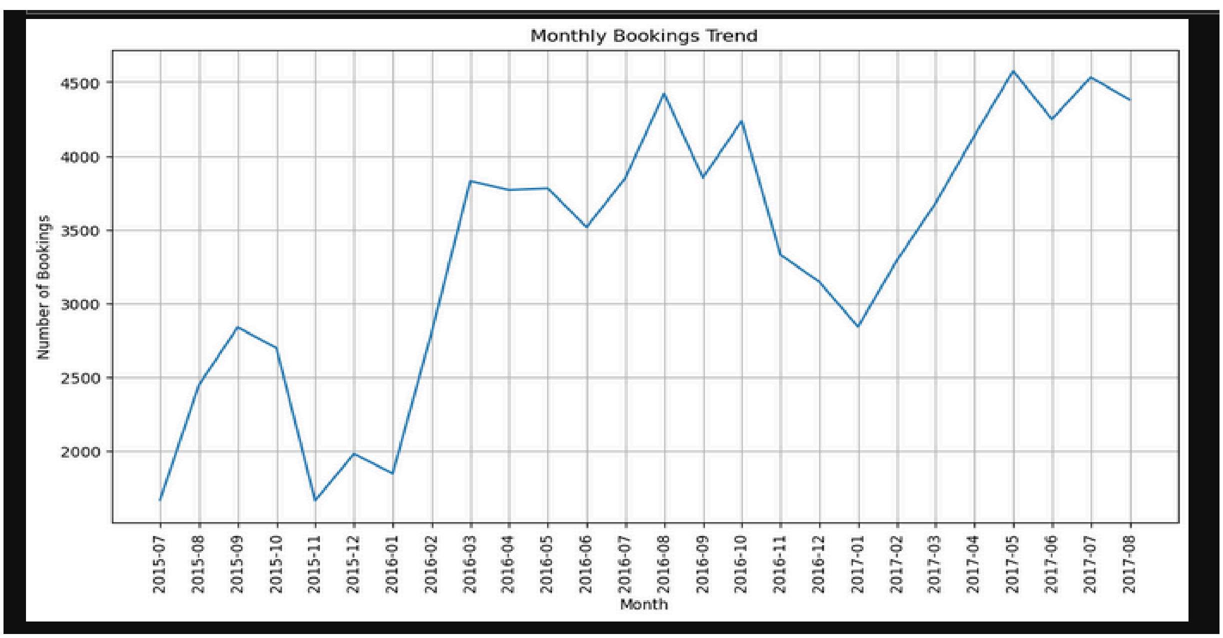
[261]:

	arrival_date	arrival_date_year	arrival_date_month	arrival_date_day_of_month
0	2015-07-01	2015	July	1
1	2015-07-01	2015	July	1
2	2015-07-01	2015	July	1
3	2015-07-01	2015	July	1
4	2015-07-01	2015	July	1

[262]:
```python
# Group by year and month to see the number of bookings
monthly_bookings = new_df.groupby(new_df['arrival_date'].dt.to_period('M')).size()

# Convert movie bookings to a dataframe
monthly_bookings = monthly_bookings.to_frame(name='bookings').reset_index()
```

[263]:
```python
# Plot the monthly bookings
plt.figure(figsize=(12, 6))
plt.plot(monthly_bookings['arrival_date'].astype(str), monthly_bookings['bookings'])
plt.xlabel('Month')
plt.ylabel('Number of Bookings')
plt.title('Monthly Bookings Trend')
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```



7. Feature Engineering

a) Create new features based on existing columns, such as calculating the duration of stay from stays_in_weekend_nights and stays_in_week_nights.

b) Derive new features from date columns like arrival_date_year, arrival_date_month, arrival_date_week_number, and arrival_date_day_of_month (e.g., season, month name, day of the week).

Feature Engineering

[256]:
```
# Calculate the duration of stay
data_clean['total_stay_duration'] = data_clean['stays_in_weekend_nights'] + data_clean['stays_in_week_nights']
```

[258]:
```
data_clean[['stays_in_weekend_nights', 'stays_in_week_nights', 'total_stay_duration']].tail() # Check
```

[258]:

	stays_in_weekend_nights	stays_in_week_nights	total_stay_duration
119385	2	5	7
119386	2	5	7
119387	2	5	7
119388	2	5	7
119389	2	7	9

[268]:
```
# Combining separate date values as one
data_clean['arrival_date'] = pd.to_datetime(data_clean['arrival_date_year'].astype(str) + '-' + data_clean['arrival_date_month'].ast
```

[271]:
```
# Deriving new features like: season, month name, and day of the week
data_clean['season'] = data_clean['arrival_date'].dt.quarter
data_clean['month_name'] = data_clean['arrival_date'].dt.month_name()
data_clean['day_of_week'] = data_clean['arrival_date'].dt.day_name()
```

[272]:
```
data_clean[['arrival_date', 'season', 'month_name', 'day_of_week']]
```

[272]:

	arrival_date	season	month_name	day_of_week
0	2015-07-01	3	July	Wednesday
1	2015-07-01	3	July	Wednesday
2	2015-07-01	3	July	Wednesday
3	2015-07-01	3	July	Wednesday
4	2015-07-01	3	July	Wednesday
...	...	...	...	...
119385	2017-08-30	3	August	Wednesday
119386	2017-08-31	3	August	Thursday
119387	2017-08-31	3	August	Thursday
119388	2017-08-31	3	August	Thursday

8. Handling Datetime Columns

a) Convert the reservation_status_date column to datetime format if necessary.

b) Extract additional features from the datetime column, such as day of the week, month, or hour.

Handling Datetime Columns

[279]:
```python
# Convert the reservation_status_date column to datetime.
data_clean['reservation_status_date'] = pd.to_datetime(data_clean['reservation_status_date'])
```

[281]:
```python
# Reservation Day of the Week
data_clean['reservation_day_of_week'] = data_clean['reservation_status_date'].dt.day_name()
```

[282]:
```python
# Reservation Month
data_clean['reservation_month'] = data_clean['reservation_status_date'].dt.month_name()
```

[285]:
```python
data_clean[['reservation_status_date', 'reservation_day_of_week', 'reservation_month']]
```

[285]:

	reservation_status_date	reservation_day_of_week	reservation_month
0	2015-01-07	Wednesday	January
1	2015-01-07	Wednesday	January
2	2015-02-07	Saturday	February
3	2015-02-07	Saturday	February
4	2015-03-07	Saturday	March
...	...	...	...
119385	2017-06-09	Friday	June
119386	2017-07-09	Sunday	July
119387	2017-07-09	Sunday	July
119388	2017-07-09	Sunday	July
119389	2017-07-09	Sunday	July

87370 rows × 3 columns