Hotel Booking Cancallation Prediction

Load Data

Load Hotel_Booking/hotel_bookings.csv file provided on Brightspace.

In [486…

```
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import metrics
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
```

importing the dataframe:

In [487…

```
fulldata = pd.read_csv('./hotel_bookings.csv')
fulldata.head(10)
```

Out[487…

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_wee
0	Resort Hotel	0	342	2015	July	
1	Resort Hotel	0	737	2015	July	
2	Resort Hotel	0	7	2015	July	
3	Resort Hotel	0	13	2015	July	
4	Resort Hotel	0	14	2015	July	
5	Resort Hotel	0	14	2015	July	
6	Resort Hotel	0	0	2015	July	
7	Resort Hotel	0	9	2015	July	
8	Resort Hotel	1	85	2015	July	
9	Resort Hotel	1	75	2015	July	

10 rows × 32 columns

In [488…

```
fulldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           119390 non-null  object 
 1   is_canceled                     119390 non-null  int64  
 2   lead_time                       119390 non-null  int64  
 3   arrival_date_year               119390 non-null  int64  
 4   arrival_date_month              119390 non-null  object 
 5   arrival_date_week_number        119390 non-null  int64  
 6   arrival_date_day_of_month       119390 non-null  int64  
 7   stays_in_weekend_nights         119390 non-null  int64  
 8   stays_in_week_nights            119390 non-null  int64  
 9   adults                          119390 non-null  int64  
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64  
 12  meal                            119390 non-null  object 
 13  country                         118902 non-null  object 
 14  market_segment                  119390 non-null  object 
 15  distribution_channel            119390 non-null  object 
 16  is_repeated_guest               119390 non-null  int64  
 17  previous_cancellations          119390 non-null  int64  
 18  previous_bookings_not_canceled  119390 non-null  int64  
 19  reserved_room_type              119390 non-null  object 
 20  assigned_room_type              119390 non-null  object 
 21  booking_changes                 119390 non-null  int64  
 22  deposit_type                    119390 non-null  object 
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64  
 26  customer_type                   119390 non-null  object 
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64  
 29  total_of_special_requests       119390 non-null  int64  
 30  reservation_status              119390 non-null  object 
 31  reservation_status_date         119390 non-null  object 
dtypes: float64(4), int64(16), object(12)
memory usage: 29.1+ MB
```

In [489…

```
fulldata.shape
```

Out[489…

```
(119390, 32)
```

In [490…

```
fulldata.columns
```

Out[490…

```
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')
```

All the information about the full data before we begin analysis.

1. Data Pre-processing (25%)

Drop irrelevant columns

It will significantly reduce the time and effort you need to invest. As a general guideline, columns containing IDs, dates, or irrelevant information are typically considered redundant and offer little value for predictive analysis.

dropping irrelevent columns:

In [491…

```
fulldata.drop(columns=['arrival_date_year', 'arrival_date_month', 'arrival_date_
fulldata
```

Out[491…

	hotel	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights	a
0	Resort Hotel	0	342	0	0	
1	Resort Hotel	0	737	0	0	
2	Resort Hotel	0	7	0	1	
3	Resort Hotel	0	13	0	1	
4	Resort Hotel	0	14	0	2	
...	...	...	...	...	...	
119385	City Hotel	0	23	2	5	
119386	City Hotel	0	102	2	5	
119387	City Hotel	0	34	2	5	
119388	City Hotel	0	109	2	5	
119389	City Hotel	0	205	2	7	

119390 rows × 24 columns

Reasons for dropping columns:

('arrival_date_year', 'arrival_date_month', 'arrival_date_week_number', 'arrival_date_day_of_month') -- dates are irrelevent for predicting return..('country') -- this info could lead to discrimination and is not useful..('reservation_status') -- might not provide meaningful predictive power compared to other features..('market_segment') -- the distribution_channel contain all the information and more so this coulmn is useless.. ('reservation_status_date') -- already dropped the reservation_status column so this column is no use to me..

1.1 Missing Values (10%)

Identify and handle missing values.

identifying missing information:

In [492…
```
fulldata.isnull().sum()
```

Out[492…
```
hotel                                 0
is_canceled                           0
lead_time                             0
stays_in_weekend_nights               0
stays_in_week_nights                  0
adults                                0
children                              4
babies                                0
meal                                  0
distribution_channel                  0
is_repeated_guest                     0
previous_cancellations                0
previous_bookings_not_canceled        0
reserved_room_type                    0
assigned_room_type                    0
booking_changes                       0
deposit_type                          0
agent                             16340
company                          112593
days_in_waiting_list                  0
customer_type                         0
adr                                   0
required_car_parking_spaces           0
total_of_special_requests             0
dtype: int64
```

The difference between the isna and the isnull is:

isna() is used to detect the missing values in the cells of the pandas data frame. It returns a data frame of the same size with the values masked as True for NA values and False for non-NA values.snull() is also used to identify or detect the missing values in the data frame. It is just an alias for isna() method.

In [493…
```
fulldata.drop(columns=['agent', 'company'], inplace= True)
fulldata.isnull().sum()
```

Out[493…
```
hotel                             0
is_canceled                       0
lead_time                         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          4
babies                            0
meal                              0
distribution_channel              0
is_repeated_guest                 0
previous_cancellations            0
previous_bookings_not_canceled    0
reserved_room_type                0
assigned_room_type                0
booking_changes                   0
deposit_type                      0
days_in_waiting_list              0
customer_type                     0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
dtype: int64
```

The columns 'agent' and 'company' contain alot of null values. Dropping these columns would be ideal as they contain alot of irrelevent data

In [494…
```
fulldata['children'] = fulldata['children'].fillna(0)
```

I have decided to fill the null values in the cloumn 'children' instead of dropping is because there are only 4 null values compared to the 1000+ in 'agent' and 'company'

Unique values

Find out unique values in columns. This will help you in identifying in-consistent data.

In [495…
```
fulldata.hotel.value_counts()
```

Out[495…
```
hotel
City Hotel      79330
Resort Hotel    40060
Name: count, dtype: int64
```

In [496…
```
fulldata.is_canceled.value_counts()
```

Out[496…
```
is_canceled
0    75166
1    44224
Name: count, dtype: int64
```

In [497…
```
fulldata.children.value_counts()
```

Out[497…

```
children
0.0     110800
1.0       4861
2.0       3652
3.0         76
10.0         1
Name: count, dtype: int64
```

In [498…

```
fulldata[['meal', 'distribution_channel' ]].value_counts()
```

Out[498…

```
meal       distribution_channel
BB         TA/TO                    73712
HB         TA/TO                    12625
BB         Direct                   12109
SC         TA/TO                    10132
BB         Corporate                 6370
HB         Direct                    1673
Undefined  TA/TO                      849
FB         TA/TO                      552
SC         Direct                     396
Undefined  Direct                     273
FB         Direct                     194
HB         Corporate                  164
BB         GDS                        115
SC         GDS                         78
FB         Corporate                   52
Undefined  Corporate                   47
SC         Corporate                   44
BB         Undefined                    4
HB         Undefined                    1
Name: count, dtype: int64
```

I have done the value_count() for both values at the same time. This is helpful as it displays the data together like a table

In [499…

```
fulldata.reserved_room_type.value_counts()
```

Out[499…

```
reserved_room_type
A    85994
D    19201
E     6535
F     2897
G     2094
B     1118
C      932
H      601
P       12
L        6
Name: count, dtype: int64
```

In [500…

```
fulldata.assigned_room_type.value_counts()
```

Out[500…
```
assigned_room_type
A    74053
D    25322
E     7806
F     3751
G     2553
C     2375
B     2163
H      712
I      363
K      279
P       12
L        1
Name: count, dtype: int64
```

In [501…
```
fulldata.deposit_type.value_counts()
```

Out[501…
```
deposit_type
No Deposit    104641
Non Refund     14587
Refundable       162
Name: count, dtype: int64
```

In [502…
```
fulldata.customer_type.value_counts()
```

Out[502…
```
customer_type
Transient          89613
Transient-Party    25124
Contract            4076
Group                577
Name: count, dtype: int64
```

1.2 Removing Inconsistent values and Outliers (10%)

Detecting inconsistencies can be achieved through a variety of methods. Some can be identified by examining unique values within each column, while others may require a solid understanding of the problem domain. Since you might not be an expert in the hotel or hospitality industry, here are some helpful hints:

Hints:

1. Check for incomplete bookings, such as reservations with zero adults, babies, or children.
2. Examine rows with zeros in both 'stays_in_weekend_nights' and 'stays_in_week_nights.'

Checking for incomplete bookings in 'adults', 'children', 'babies':

In [503…
```
fulldata[(fulldata['adults']==0) & (fulldata['children']==0) & (fulldata['babies
```

Out[503…
```
Index([  2224,   2409,   3181,   3684,   3708,   4127,   9376,  31765,  32029,
        32827,
       ...
       112558, 113188, 114583, 114908, 114911, 115029, 115091, 116251, 116534,
       117087],
      dtype='int64', length=180)
```

Examine rows with zeros:

In [504…
```
fulldata[(fulldata['stays_in_weekend_nights']==0) & (fulldata['stays_in_week_nig
```

Out[504…
```
Index([     0,      1,    167,    168,    196,    197,    459,    568,    569,
          618,
       ...
       113930, 114678, 114908, 114911, 115482, 115483, 117701, 118029, 118631,
       118963],
      dtype='int64', length=715)
```

Dropping 0 values:

In [505…
```
fulldata = fulldata.drop(fulldata[fulldata['adults']== 0].index)
```

Dropping these values makes analysing the real data easier. Null or 0 values are of no use.

Testing:

In [506…
```
fulldata[fulldata['adults']== 0].index
```

Out[506…
```
Index([], dtype='int64')
```

Dropping 0 values in columns 'stays_in_weekend'and 'stays_in_week_nights:

In [507…
```
fulldata = fulldata.drop(fulldata[(fulldata['stays_in_weekend_nights']== 0) & (f
```

Testing:

In [508…
```
fulldata[(fulldata['stays_in_weekend_nights']== 0) & (fulldata['stays_in_week_ni
```

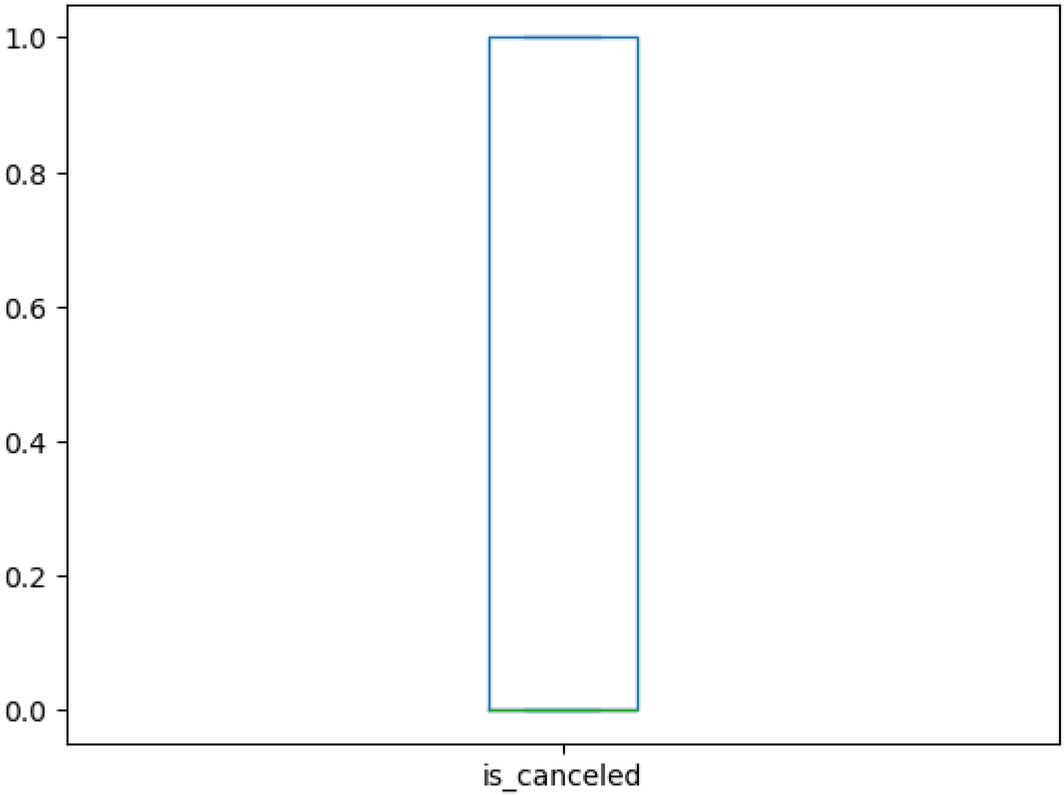
Out[508…
```
Index([], dtype='int64')
```

Outliers:

Boxplots example:

In [509…
```
fulldata.plot(y=['is_canceled'], kind='box')
```

Out[509…
```
<Axes: >
```



Boxplotting and Removing Outlier:

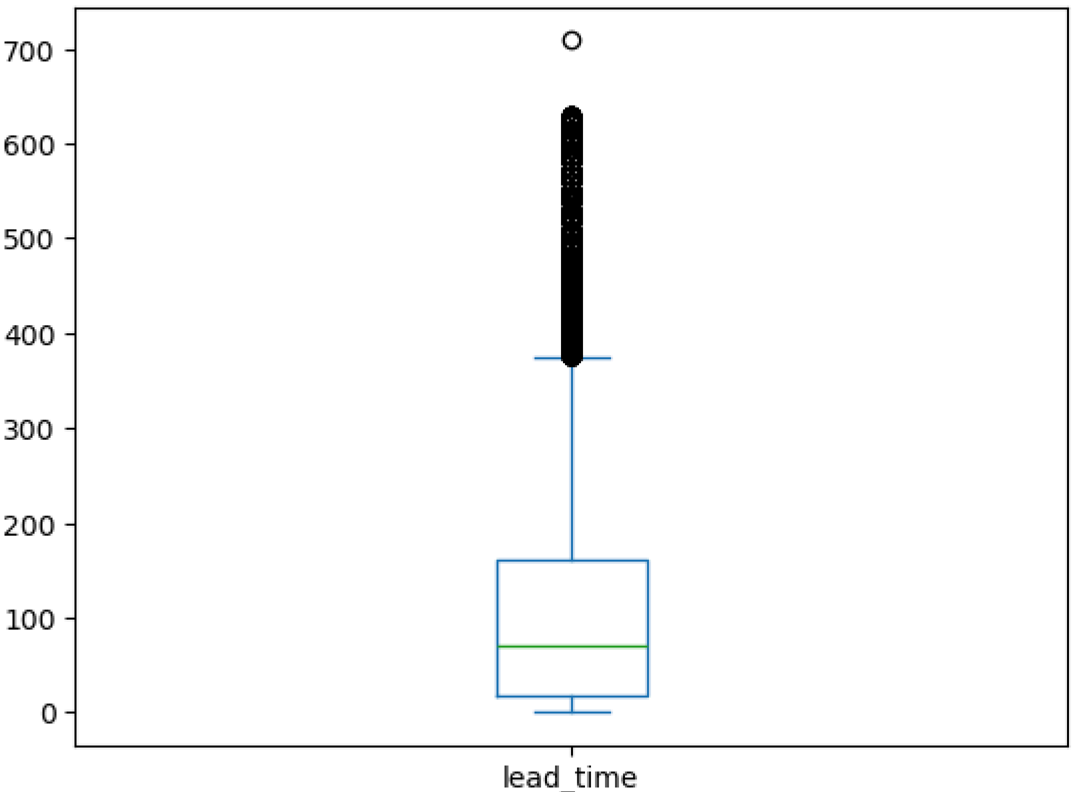
lead_time:

In [510…

```
fulldata.plot(y=['lead_time'], kind='box')
```

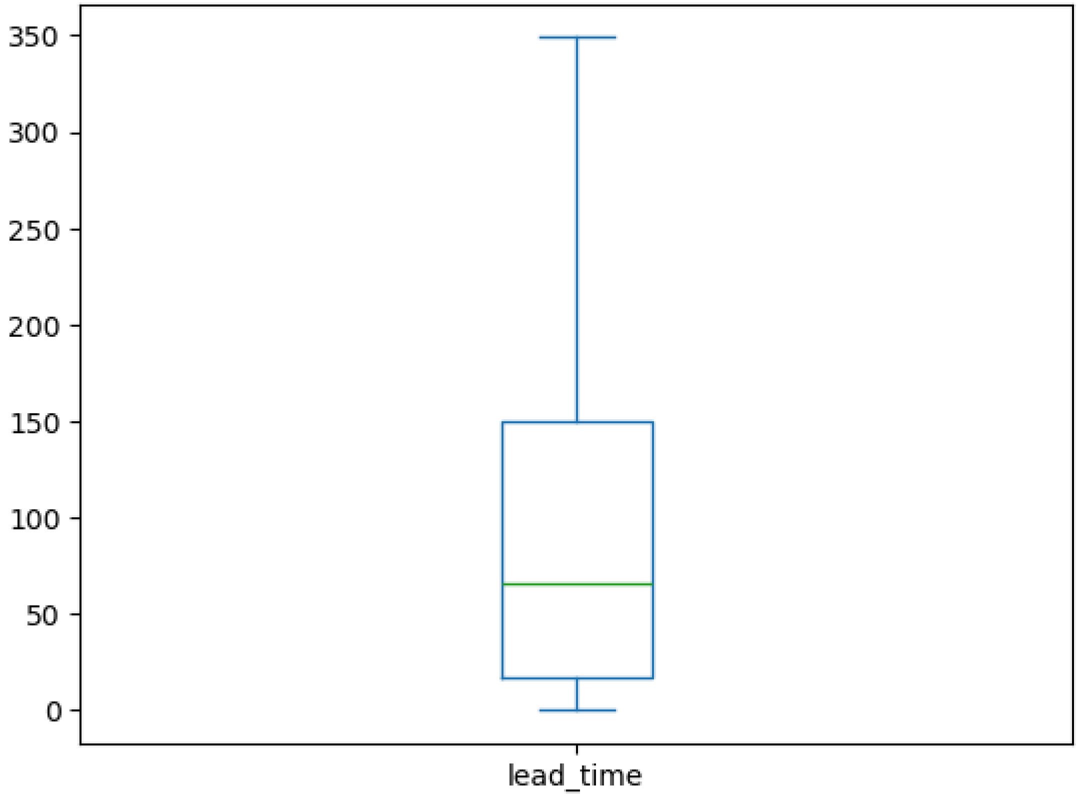
Out[510…

```
<Axes: >
```



In [511…
```
fulldata = fulldata.drop(fulldata[fulldata['lead_time']> 349].index)
fulldata.plot(y=['lead_time'], kind='box')
```

Out[511… <Axes: >



stays_in_weekend_nights:

In [512…
```
fulldata.plot(y=['stays_in_weekend_nights'], kind='box')
```

Out[512… <Axes: >

In [513…

```
fulldata = fulldata.drop(fulldata[fulldata['stays_in_weekend_nights']> 5].index)
fulldata.plot(y=['stays_in_weekend_nights'], kind='box')
```

Out[513…

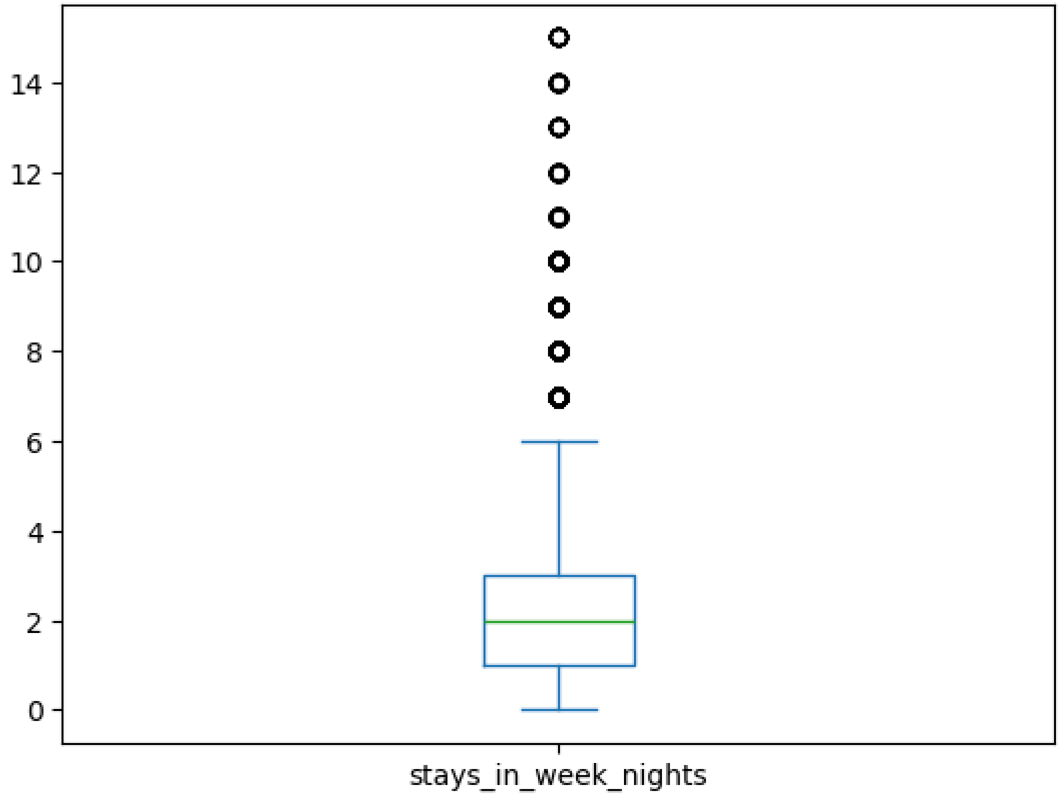
```
<Axes: >
```

stays_in_week_nights:

In [514…

```
fulldata.plot(y=['stays_in_week_nights'], kind='box')
```

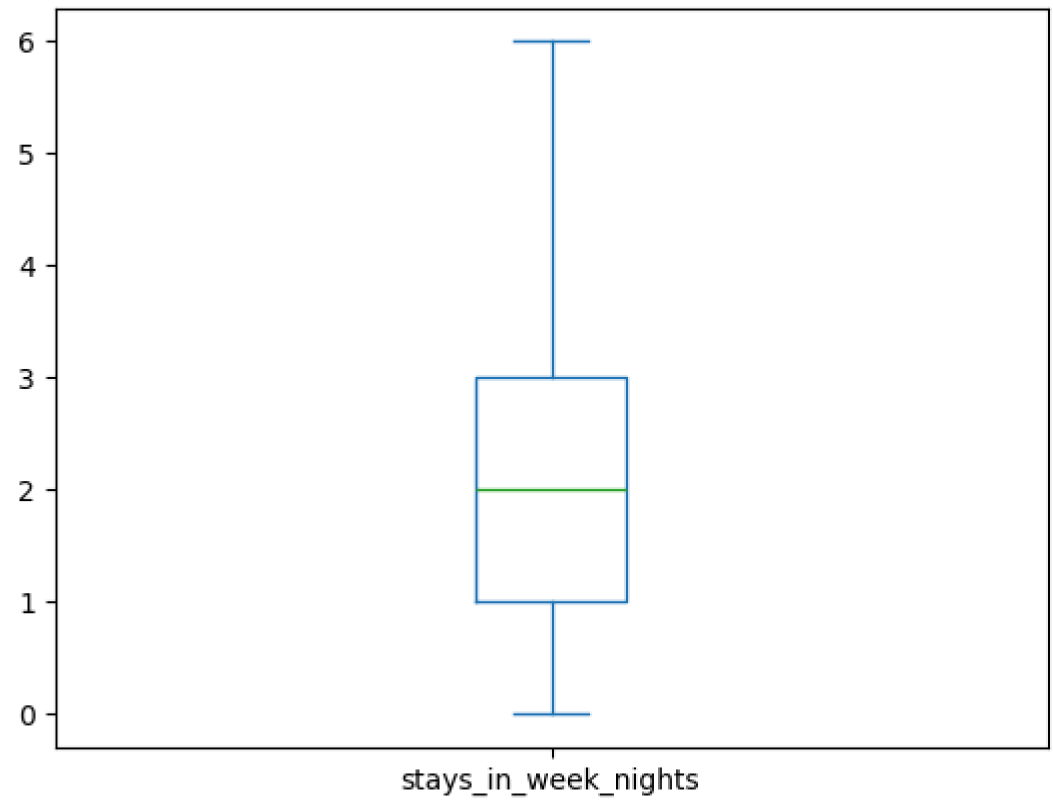
Out[514… <Axes: >



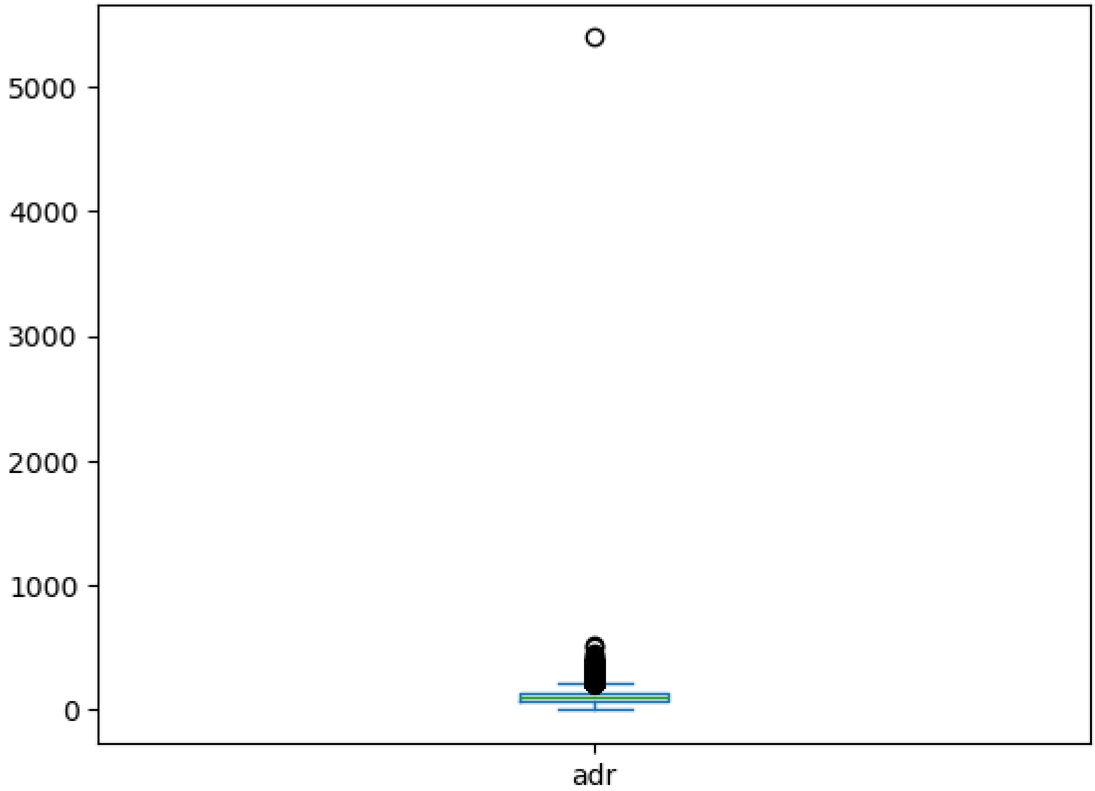
In [515…

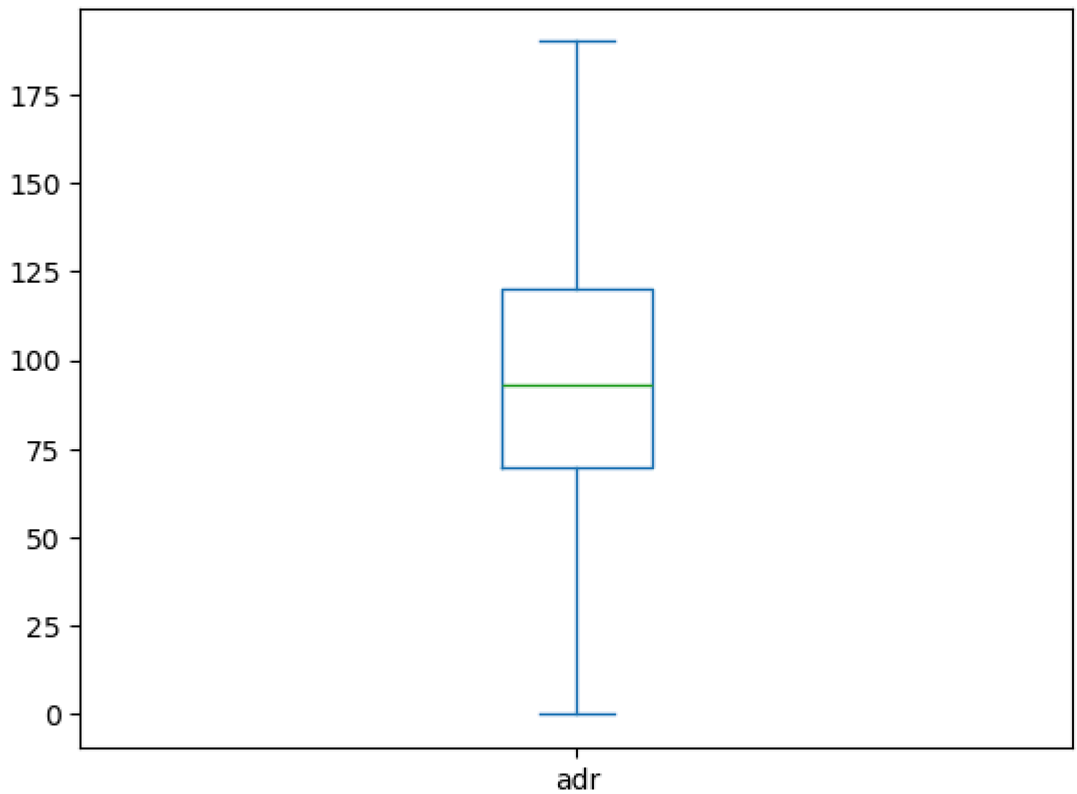
```
fulldata = fulldata.drop(fulldata[fulldata['stays_in_week_nights']> 6].index)
fulldata.plot(y=['stays_in_week_nights'], kind='box')
```

Out[515… <Axes: >



ADR:

In [516…
```
fulldata.plot(y=['adr'], kind='box')
```
Out[516…
```
<Axes: >
```


In [517…
```
fulldata = fulldata.drop(fulldata[(fulldata['adr']>190) | (fulldata['adr'] < 0)]
fulldata.plot(y=['adr'], kind='box')
```
Out[517…
```
<Axes: >
```


I have chosen to remove th outlier in only these columns and not the others because these are essential columns where the irrelevent data does not mean anything. for example deleting the outliers in the 'adult' column would mean some data is altered in the 'children and 'babies' columns to. This is because children and babies cannot make a booking.

1.3 Column data type conversion (5%)

All necessary columns should be correctly converted to appropriate data types.

In [ ]:

```
```

In [518…

```
fulldata['children'] = fulldata['children'].astype('int64')
```

2. Exploratory Data Analysis (25%)

You've also been provided with examples of valuable insights that could be of interest to hotel management, including:

- Calculating cancellation percentages for City and Resort hotels.
- Identifying the most frequently ordered meal types.
- Determining the number of returning guests.
- Discovering the most booked room types.
- Exploring correlations between room types and cancellations.

Visualize these insights using three different types of visualizations covered in the practicals, such as:

- Bar graphs
- Pie charts
- Line charts
- Heatmaps

2.1. Calculating cancellation percentages for City and Resort hotels.

In [519…

```
hotel_location = fulldata.hotel.unique()
```

In [520…

```
city_hotel_data = fulldata.drop(fulldata[fulldata['hotel']== hotel_location[0]].
resort_hotel_data = fulldata.drop(fulldata[fulldata['hotel']== hotel_location[1]
```

Splitting data into 2 different datasets

In [521…

```
city_hotel_data = city_hotel_data.is_canceled.value_counts()
resort_hotel_data = resort_hotel_data.is_canceled.value_counts()
```

Turning is_canceled column into a verible

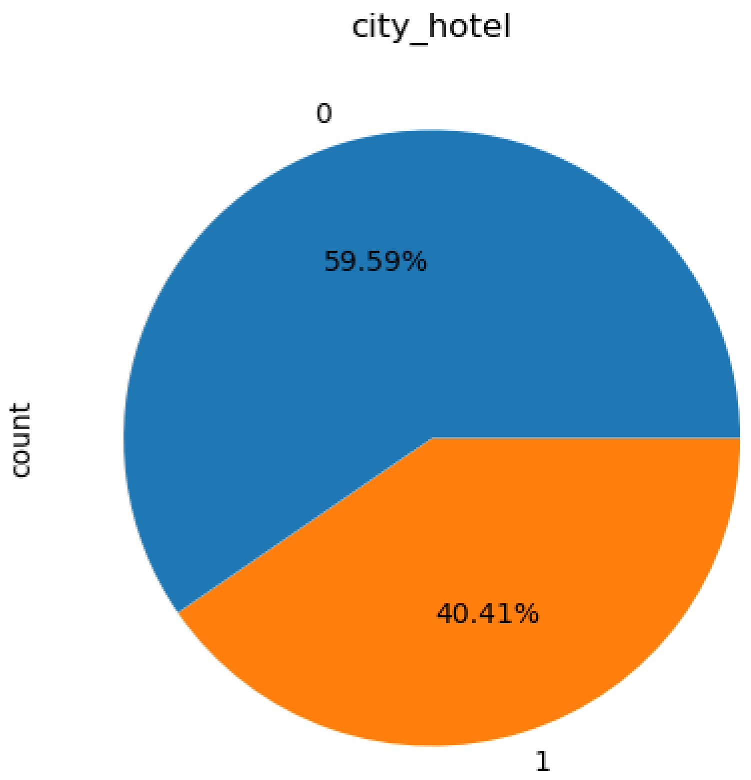
Plotting the graphs:

In [522…

```
city_hotel_data.plot(kind= 'pie', title= 'city_hotel', figsize = [9,5], autopct=
```

Out[522…

```
<Axes: title={'center': 'city_hotel'}, ylabel='count'>
```

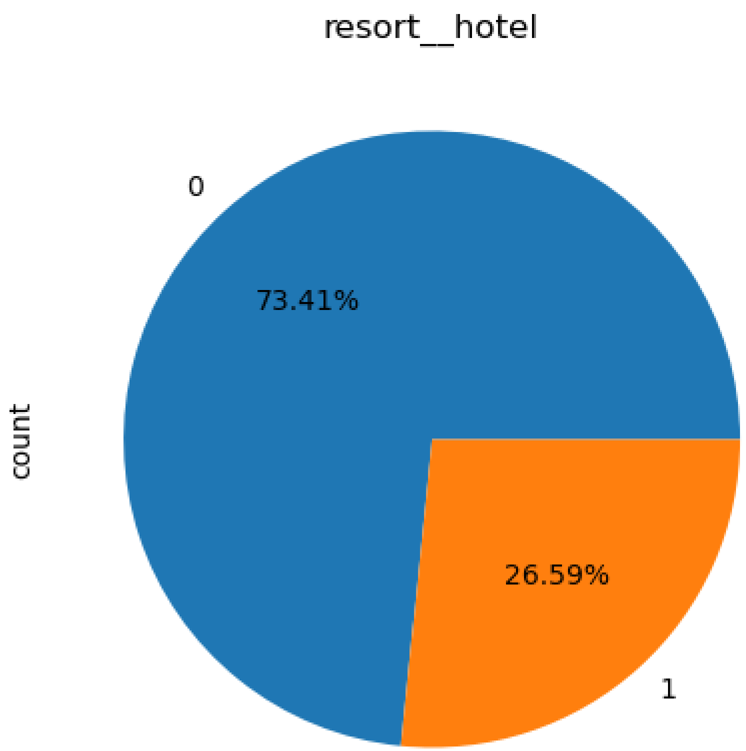


In [523…

```
resort_hotel_data.plot(kind= 'pie', title= 'resort__hotel', figsize = [9,5], aut
```

Out[523…

```
<Axes: title={'center': 'resort__hotel'}, ylabel='count'>
```



The analysis we can make from the above 2 graphs is that there are more cancellations in the 'resort hotel', having more then 14% percentage cancellations.

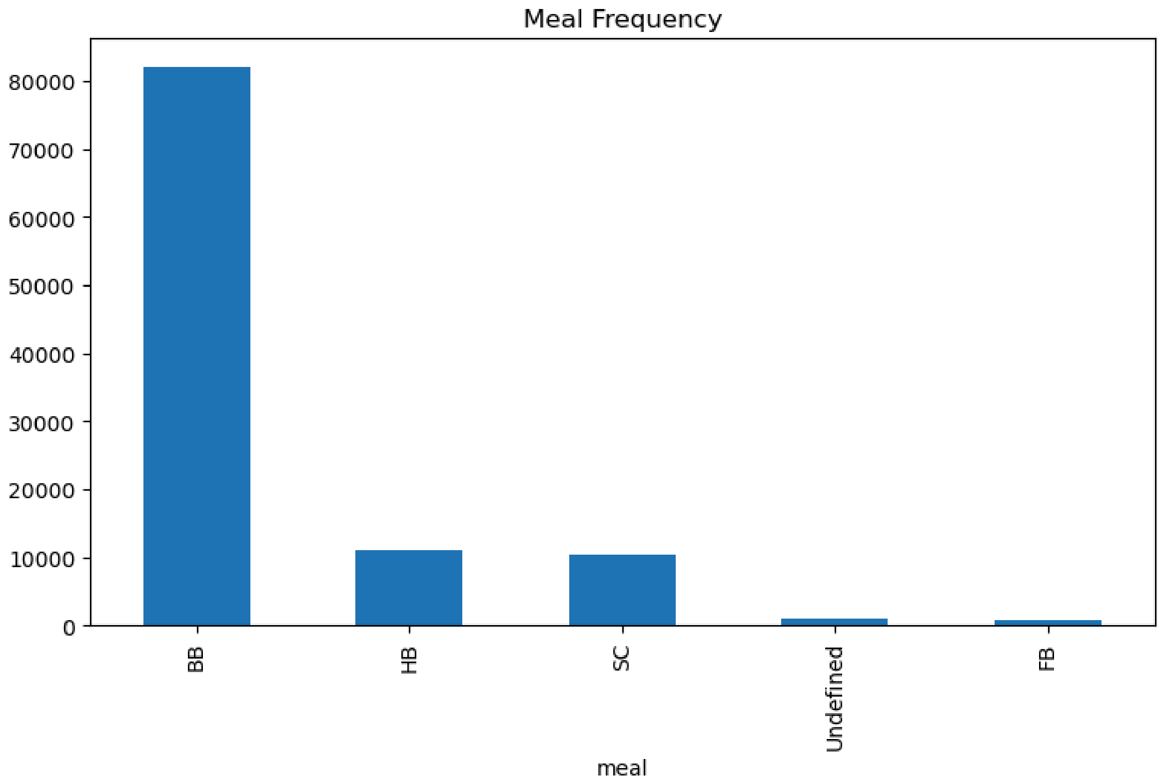
2.2. Identifying the most frequently ordered meal types.

In [524…

```
# Meal value count as a bar chart
fulldata.meal.value_counts().plot(kind='bar',title='Meal Frequency',figsize=[9,5
```

Out[524…

```
<Axes: title={'center': 'Meal Frequency'}, xlabel='meal'>
```



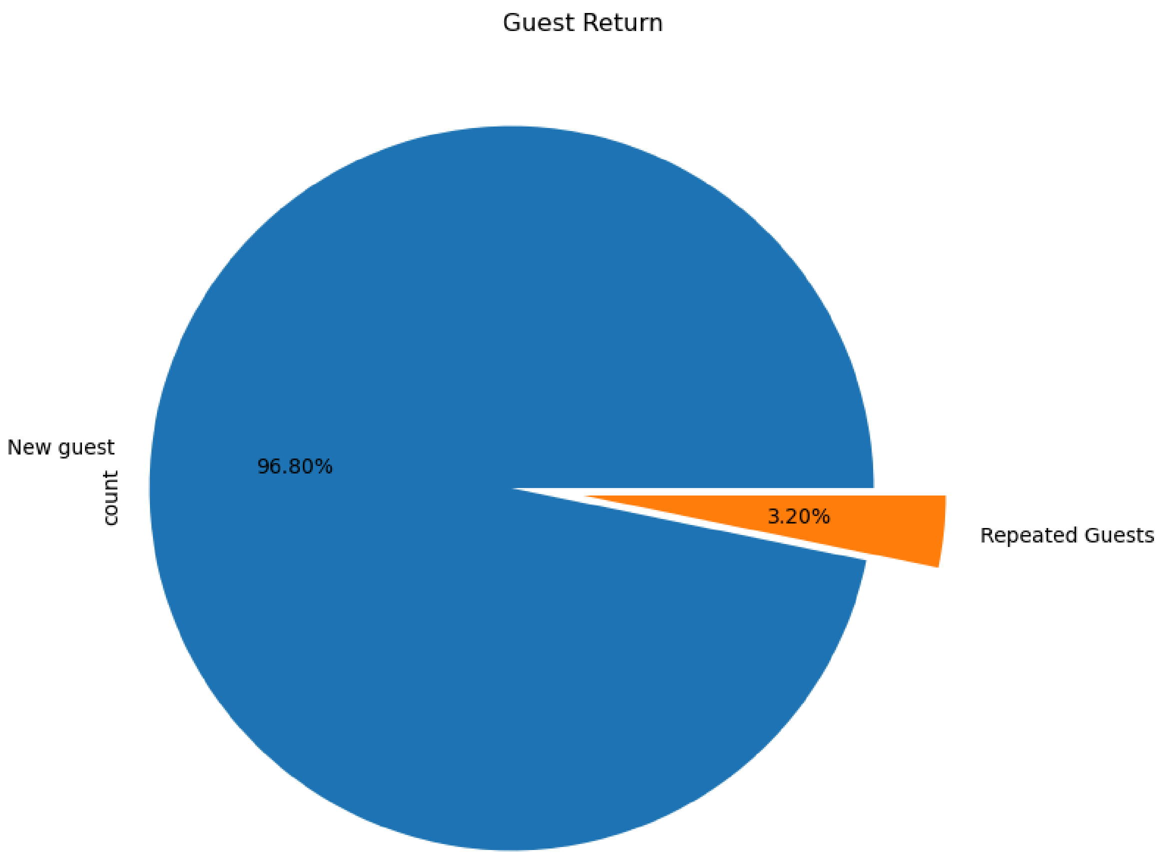
2.3. Determining the number of returning guests.

In [525…

```
explode = (0.2,0)
fulldata.is_repeated_guest.value_counts().plot(kind='pie',labels={'New guest','R
```

Out[525…

```
<Axes: title={'center': 'Guest Return'}, ylabel='count'>
```



In [526…

```
guests_repeated = fulldata.is_repeated_guest.value_counts()
guests_repeated
```

Out[526…

```
is_repeated_guest
0    101757
1      3361
Name: count, dtype: int64
```

The total number of guests that came back is 3361.

2.4. Discovering the most booked room types.

In [527…

```
room_types = fulldata.reserved_room_type.value_counts()
room_types
```

Out[527…

```
reserved_room_type
A    78972
D    16732
E     4881
F     1701
G     1092
B      869
C      558
H      308
L        5
Name: count, dtype: int64
```

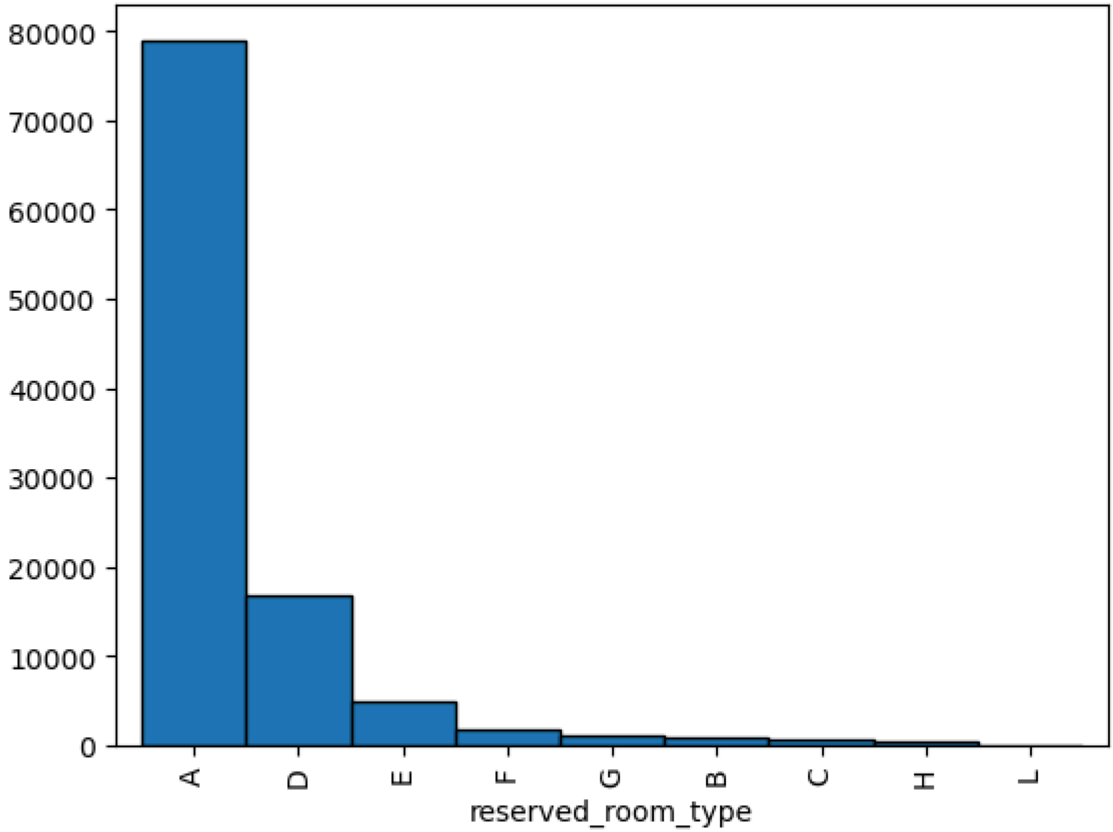
Count the occurrences of each unique value in the 'reserved_room_type' column. Finding the most reserved room type.

In [528…

```
room_types.plot(kind='bar', edgecolor = 'black', width = 1)
```

Out[528…

```
<Axes: xlabel='reserved_room_type'>
```

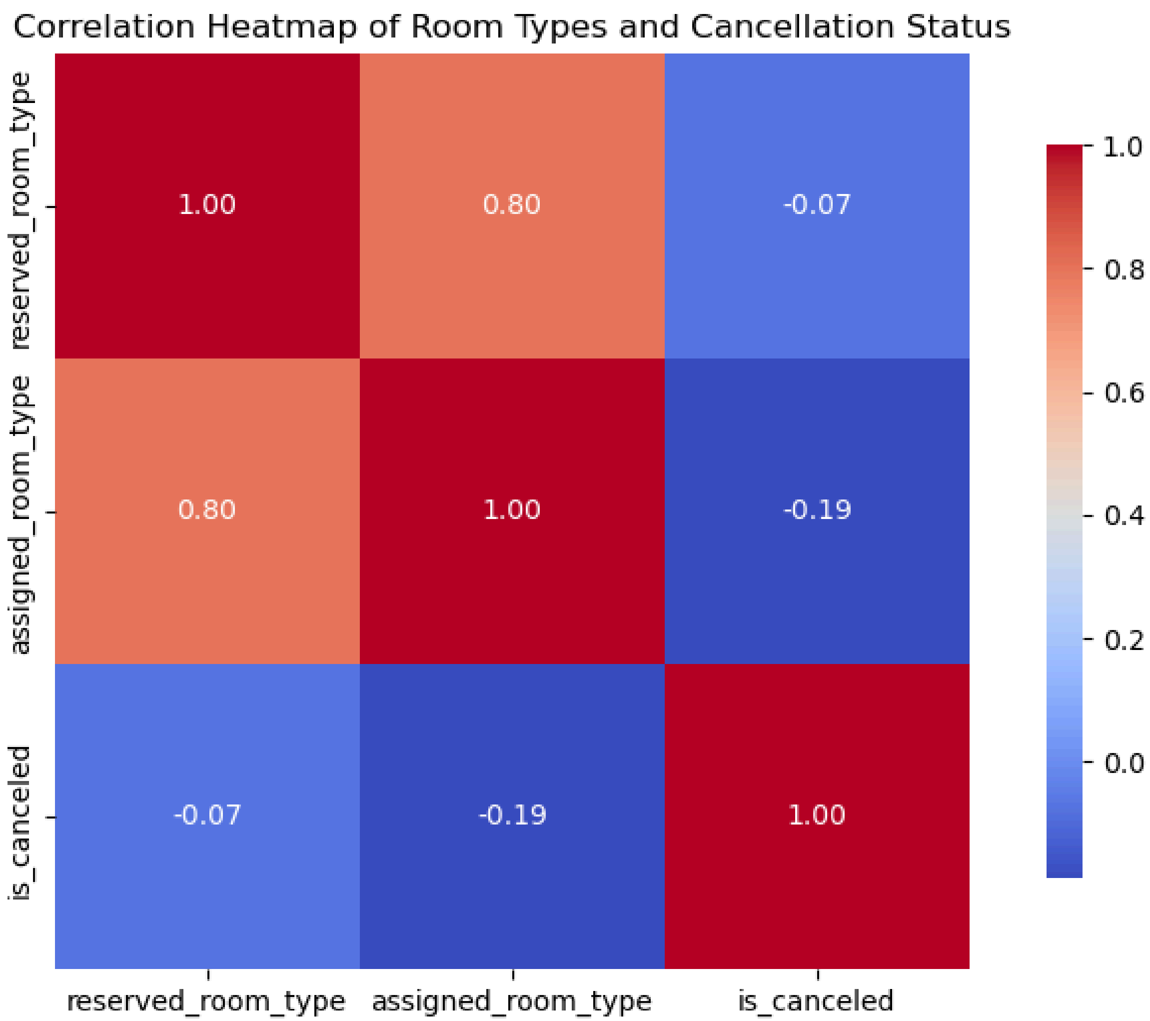


As it can be seen in the above graph room_type A is the most popular

2.5. Exploring correlations between room types and cancellations.

In [529…

```
# Convert 'reserved_room_type' and 'assigned_room_type' to categorical and get t
fulldata['reserved_room_type'] = fulldata['reserved_room_type'].astype('category
fulldata['assigned_room_type'] = fulldata['assigned_room_type'].astype('category

# Create a DataFrame containing reserved and assigned room types along with canc
canceled_rooms = fulldata[['reserved_room_type', 'assigned_room_type', 'is_cance

# Calculate the correlation matrix of the selected features
correlation = canceled_rooms.corr().round(2)

# Set up the matplotlib figure
plt.figure(figsize=(10, 6))

# Create a heatmap to visualise the correlation
sb.heatmap(correlation, cmap="coolwarm", annot=True, fmt='.2f', square=True, cba

# Set title for better understanding
plt.title('Correlation Heatmap of Room Types and Cancellation Status')
plt.show()
```



The correlation heatmap of the canceled_rooms DataFrame reveals important relationships among the variables reserved_room_type, assigned_room_type, and is_canceled.

Room Types Correlation: If the correlation between reserved_room_type and assigned_room_type is high (close to 1), it indicates that guests typically receive the room they reserved, suggesting efficient management of room assignments.

Cancellation Insights: A strong negative correlation between either room type and is_canceled suggests that certain room types are less likely to be canceled. This could indicate popularity or customer satisfaction with those specific types. Conversely, if a room type has a high positive correlation with is_canceled, it may point to issues that make that room type less desirable.

3. Feature Engineering (20%)

Apply various feature engineering techniques, covered in the lectures and practicals.

Hint:

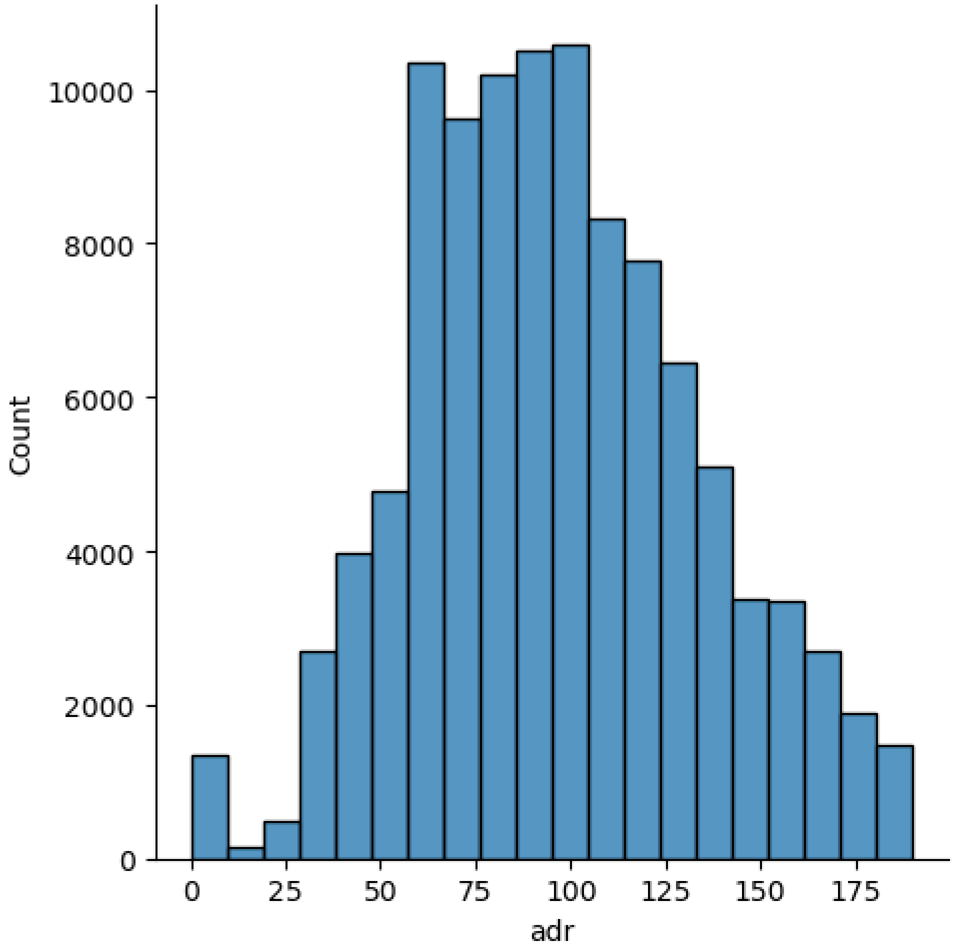
- Binning

- Encoding
- Scaling
- Feature selection

3.1. Binning

In [530…
```
#Create a distribution plot for the 'adr' (average daily rate) variable
sb.displot(data = fulldata, x = 'adr', bins = 20)
```

Out[530… `<seaborn.axisgrid.FacetGrid at 0x26c681eb350>`



Binning reduces the noise in the data and makes it easier to visualise and analyse trends. It allows you to summarise large datasets by organizing data into intervals. As it can be seen in the above graph using bins narrows the data making it easier to assess.

3.2. Encoding

Hotel column:

In [531…
```
fulldata['hotel'] = fulldata['hotel'].astype('category').cat.codes
```

Access the interget values in the category data

In [532…
```
fulldata.reset_index(drop=True,inplace=True)
```

Relabing the indexs

Meal column:

In [533…

```
ohe = OneHotEncoder(sparse_output= False)

#applying One Hot coding to the meal column:
ohe_coded = ohe.fit_transform(fulldata[['meal']])

#converting the resutl into Dataframe:
one_hot = pd.DataFrame(ohe_coded, columns= ohe.get_feature_names_out(['meal']))

#drop original column
fulldata = fulldata.drop('meal', axis = 1)

#join the new One Hot code to the original
fulldata = fulldata.join(one_hot)

fulldata
```

Out[533…

	hotel	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights	a
0	1	0	7	0	1	
1	1	0	13	0	1	
2	1	0	14	0	2	
3	1	0	14	0	2	
4	1	0	0	0	2	
...	...	...	...	...	...	
105113	0	0	164	2	4	
105114	0	0	21	2	5	
105115	0	0	23	2	5	
105116	0	0	34	2	5	
105117	0	0	109	2	5	

105118 rows × 26 columns

Distribution_channel:

In [534…

```
#applying One Hot coding to the distribution_channel column:
ohe_coded = ohe.fit_transform(fulldata[['distribution_channel']])

#converting the resutl into Dataframe:
one_hot = pd.DataFrame(ohe_coded, columns= ohe.get_feature_names_out(['distribut

#drop original column
fulldata = fulldata.drop('distribution_channel', axis = 1)

#join the new One Hot code to the original
```

```
fulldata = fulldata.join(one_hot)

fulldata
```

Out[534…

	hotel	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights	a
0	1	0	7	0	1	
1	1	0	13	0	1	
2	1	0	14	0	2	
3	1	0	14	0	2	
4	1	0	0	0	2	
...	...	...	...	...	...	
105113	0	0	164	2	4	
105114	0	0	21	2	5	
105115	0	0	23	2	5	
105116	0	0	34	2	5	
105117	0	0	109	2	5	

105118 rows × 30 columns

Deposit_type:

In [535…

```
#using the map() function to manually convert the data:
fulldata['deposit_type'] = fulldata['deposit_type'].map({deposit_type[0]: 0,depo

fulldata.deposit_type.value_counts()
```

Out[535…

```
deposit_type
0    92781
2    12186
1      151
Name: count, dtype: int64
```

Using the map() function is the same as replace().

In [536…

```
# Checking to see all the columns are no longer have object datatype:
fulldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105118 entries, 0 to 105117
Data columns (total 30 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           105118 non-null  int8   
 1   is_canceled                     105118 non-null  int64  
 2   lead_time                       105118 non-null  int64  
 3   stays_in_weekend_nights         105118 non-null  int64  
 4   stays_in_week_nights            105118 non-null  int64  
 5   adults                          105118 non-null  int64  
 6   children                        105118 non-null  int64  
 7   babies                          105118 non-null  int64  
 8   is_repeated_guest               105118 non-null  int64  
 9   previous_cancellations          105118 non-null  int64  
 10  previous_bookings_not_canceled  105118 non-null  int64  
 11  reserved_room_type              105118 non-null  int8   
 12  assigned_room_type              105118 non-null  int8   
 13  booking_changes                 105118 non-null  int64  
 14  deposit_type                    105118 non-null  int32  
 15  days_in_waiting_list            105118 non-null  int64  
 16  customer_type                   105118 non-null  object 
 17  adr                             105118 non-null  float64
 18  required_car_parking_spaces     105118 non-null  int64  
 19  total_of_special_requests       105118 non-null  int64  
 20  meal_BB                         105118 non-null  float64
 21  meal_FB                         105118 non-null  float64
 22  meal_HB                         105118 non-null  float64
 23  meal_SC                         105118 non-null  float64
 24  meal_Undefined                  105118 non-null  float64
 25  distribution_channel_Corporate  105118 non-null  float64
 26  distribution_channel_Direct     105118 non-null  float64
 27  distribution_channel_GDS        105118 non-null  float64
 28  distribution_channel_TA/TO      105118 non-null  float64
 29  distribution_channel_Undefined  105118 non-null  float64
dtypes: float64(11), int32(1), int64(14), int8(3), object(1)
memory usage: 21.6+ MB
```

In [537…

```
#applying One Hot coding to the customer_type column:
ohe_coded = ohe.fit_transform(fulldata[['customer_type']])

#converting the resutl into Dataframe:
one_hot = pd.DataFrame(ohe_coded, columns= ohe.get_feature_names_out(['customer_

#drop original column
fulldata = fulldata.drop('customer_type', axis = 1)

#join the new One Hot code to the original
fulldata = fulldata.join(one_hot)

fulldata
```

Out[537…

	hotel	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights	a
0	1	0	7	0	1	
1	1	0	13	0	1	
2	1	0	14	0	2	
3	1	0	14	0	2	
4	1	0	0	0	2	
...	...	...	...	...	...	
105113	0	0	164	2	4	
105114	0	0	21	2	5	
105115	0	0	23	2	5	
105116	0	0	34	2	5	
105117	0	0	109	2	5	

105118 rows × 33 columns

In [538…

```
fulldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105118 entries, 0 to 105117
Data columns (total 33 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           105118 non-null  int8   
 1   is_canceled                     105118 non-null  int64  
 2   lead_time                       105118 non-null  int64  
 3   stays_in_weekend_nights         105118 non-null  int64  
 4   stays_in_week_nights            105118 non-null  int64  
 5   adults                          105118 non-null  int64  
 6   children                        105118 non-null  int64  
 7   babies                          105118 non-null  int64  
 8   is_repeated_guest               105118 non-null  int64  
 9   previous_cancellations          105118 non-null  int64  
 10  previous_bookings_not_canceled  105118 non-null  int64  
 11  reserved_room_type              105118 non-null  int8   
 12  assigned_room_type              105118 non-null  int8   
 13  booking_changes                 105118 non-null  int64  
 14  deposit_type                    105118 non-null  int32  
 15  days_in_waiting_list            105118 non-null  int64  
 16  adr                             105118 non-null  float64
 17  required_car_parking_spaces     105118 non-null  int64  
 18  total_of_special_requests       105118 non-null  int64  
 19  meal_BB                         105118 non-null  float64
 20  meal_FB                         105118 non-null  float64
 21  meal_HB                         105118 non-null  float64
 22  meal_SC                         105118 non-null  float64
 23  meal_Undefined                  105118 non-null  float64
 24  distribution_channel_Corporate  105118 non-null  float64
 25  distribution_channel_Direct     105118 non-null  float64
 26  distribution_channel_GDS        105118 non-null  float64
 27  distribution_channel_TA/TO      105118 non-null  float64
 28  distribution_channel_Undefined  105118 non-null  float64
 29  customer_type_Contract          105118 non-null  float64
 30  customer_type_Group             105118 non-null  float64
 31  customer_type_Transient         105118 non-null  float64
 32  customer_type_Transient-Party   105118 non-null  float64
dtypes: float64(15), int32(1), int64(14), int8(3)
memory usage: 24.0 MB
```

Conclusion:

Converted the data into categorical using encoding to get a better understanding of the the values and allows me to make amendments to the indivisual values.

3.3. Scaling

In [539…

```
# Initialising the StandardScaler
std_scaler = StandardScaler()

# Fit the scaler to the data and transform it, creating a new DataFrame with sta
fulldata_std = pd.DataFrame(std_scaler.fit_transform(fulldata.values), columns=f

print("Dataset using Standard Scaling")

fulldata_std
```

```
Dataset using Standard Scaling
```

Out[539…

	hotel	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_night
0	1.486312	-0.751681	-0.955664	-0.987359	-0.91939
1	1.486312	-0.751681	-0.889210	-0.987359	-0.91939
2	1.486312	-0.751681	-0.878134	-0.987359	-0.21097
3	1.486312	-0.751681	-0.878134	-0.987359	-0.21097
4	1.486312	-0.751681	-1.033194	-0.987359	-0.21097
...	...	...	...	...	..
105113	-0.672806	-0.751681	0.783221	1.316253	1.205851
105114	-0.672806	-0.751681	-0.800604	1.316253	1.91426
105115	-0.672806	-0.751681	-0.778453	1.316253	1.91426
105116	-0.672806	-0.751681	-0.656620	1.316253	1.91426
105117	-0.672806	-0.751681	0.174058	1.316253	1.91426

105118 rows × 33 columns

In this code, 'StandardScaler' standardises the features of 'fulldata' by removing the mean and scaling to unit variance. The transformed data is stored in 'fulldata_std', which maintains the original column names and indices for clarity. This process helps to improve the performance of machine learning algorithms by ensuring that all features contribute equally.

3.4. Feature selection

In [540…

```
x = fulldata_std.drop(columns=['is_canceled'])
y = fulldata.is_canceled
```

The code separates the feature matrix 'X' by dropping the target variable 'is_canceled' from the standardized dataset 'fulldata_std'. It assigns the target variable 'y' to the 'is_canceled' column in the original dataset 'fulldata'.

4. Classifier Training (20%)

Utilise the sklearn Python library to train a ML model (e.g.decision tree classifier). Your process should start with splitting your dataset into input features (X) and a target feature (y). Next, divide the data into 70% training and 30% testing subsets. Train your model on the training dataset and evaluate using test dataset with appropriate metrics. Aim to achieve higher accuracy e.g. more than 70% accuracy using your model.

4.1. Data Splitting (5%)

In [541...

```
# Splitting the datasets into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3)
```

This code splits the feature matrix 'x' and target variable 'y' into training and testing sets, with 70% of the data used for training ('X_train', 'Y_train') and 30% for testing ('X_test', 'Y_test'). This helps evaluate model performance on unseen data.

4.2. Model Training (10%)

In [542...

```
# Initialize a Decision Tree Classifier with 'entropy' as the criterion and a fi
data_training = DecisionTreeClassifier(criterion='entropy', random_state=1)

# Fit the model on the training data and make predictions on the test data
data_training = data_training.fit(X_train, Y_train)
Y_prediction = data_training.predict(X_test)
```

A Decision Tree Classifier is created and trained using the training data (X_train, Y_train), and then it predicts the target variable for the test set (X_test).

4.3. Model Evaluation (5%)

In [543...

```
# Calculate the accuracy of the model by comparing predicted values to actual va
data_accuracy = metrics.accuracy_score(Y_test, Y_prediction)

# Print the accuracy score of the model
print("STD Accuracy:", data_accuracy)
```

```
STD Accuracy: 0.8138001014713343
```

Overall accurancy of the data is: *81.49%*

5. Feature Importance (10%)

Assess the importance of features within your decision tree model. Provide commentary on the reliability of your model's results based on the feature importance scores.

In [544...

```
# Print a header for the importance of features
print("Importance Classifier")

# Create a DataFrame that combines feature names with their importance scores, s
pd.concat(
    (pd.DataFrame(fulldata_std.iloc[:, 1:].columns, columns=['column:']),
     pd.DataFrame(data_training.feature_importances_, columns=['importance'])),
    axis=1).sort_values(by='importance', ascending=False)[:20]
```

```
Importance Classifier
```

Out[544…

	column:	importance
1	lead_time	0.229173
15	adr	0.201912
13	deposit_type	0.197053
3	stays_in_week_nights	0.061136
2	stays_in_weekend_nights	0.038643
17	total_of_special_requests	0.031155
8	previous_cancellations	0.030197
11	assigned_room_type	0.027573
16	required_car_parking_spaces	0.025529
12	booking_changes	0.020965
30	customer_type_Transient	0.020052
10	reserved_room_type	0.018144
4	adults	0.017801
26	distribution_channel_TA/TO	0.013084
0	is_canceled	0.011329
9	previous_bookings_not_canceled	0.009254
21	meal_SC	0.007225
5	children	0.006787
18	meal_BB	0.006301
31	customer_type_Transient-Party	0.005493

The code combines the feature names and their corresponding importance scores from the trained Decision Tree model into a single DataFrame. It then sorts this DataFrame by importance and displays the top 20 most important features, highlighting their relevance in the classification task.