Hotel Booking Cancallation Prediction

Load Data

Load Hotel_Booking/hotel_bookings.csv file provided on Brightspace.

In [360…

```
# Importing Libraries
import seaborn as sb
import pandas as pd
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 StandardScaler
import math


bookings_data = pd.read_csv('hotel_bookings.csv')
```

In [361…

```
bookings_data.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
```

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.

In [362…
```
bookings_data = bookings_data.drop(columns=
```

```
                                        ['country',
                                         'reservation_status_date',
                                         'reservation_status',
                                         'arrival_date_day_of_month',
                                         'arrival_date_week_number',
                                         'arrival_date_year',
                                         'arrival_date_month',
                                         'previous_bookings_not_canceled',
                                         'distribution_channel'],axis=1)
```

Column Drop Explanation:

- Country: Minimal predictive value.
- Reservation Status Date: Redundant date information.
- Reservation Status: Redundant date information.
- Arrival Date Day of Month: Redundant date information.
- Arrival Date Week Number: Redundant date information.
- Arrival Date Year: Redundant date information.
- Arrival Date Month: Redundant date information.
- Previous Bookings Not Canceled: Redundant; covered by existing canceled bookings column.
- Distribution Channel: Offers little additional value; similar information provided by 'market_segment'.

1.1 Missing Values (10%)

Identify and handle missing values.

In [363…

```
# Getting information of missing values in the data table:

bookings_data.isna().sum()
```

Out[363...

```
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
market_segment                     0
is_repeated_guest                  0
previous_cancellations             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
```

Company and Agent Drop:

Both columns are ID columns additionally both of them contain large amount of Null values

In [364...

```
bookings_data = bookings_data.drop(columns=['company', 'agent'],axis=1)
```

Children column fill:

In [365...

```
# We are left with the children column only with some null values, we wil
bookings_data['children'] = bookings_data['children'].fillna(0)
```

Unique values

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

In [366...

```
# Identifying diffrent hotels in the dataset:
bookings_data.hotel.unique()
```

Out[366...

```
array(['Resort Hotel', 'City Hotel'], dtype=object)
```

In [367...

```
# Identifying different types of meal:
bookings_data.meal.value_counts()
```

Out[367…
```
meal
BB           92310
HB           14463
SC           10650
Undefined     1169
FB             798
Name: count, dtype: int64
```

In [368…
```
# Finding all the market segments
bookings_data.market_segment.value_counts()
```

Out[368…
```
market_segment
Online TA        56477
Offline TA/TO    24219
Groups           19811
Direct           12606
Corporate         5295
Complementary      743
Aviation           237
Undefined            2
Name: count, dtype: int64
```

In [369…
```
# All deposit types:
bookings_data.deposit_type.value_counts()
```

Out[369…
```
deposit_type
No Deposit    104641
Non Refund     14587
Refundable       162
Name: count, dtype: int64
```

In [370…
```
# Types of customers:
bookings_data.customer_type.value_counts()
```

Out[370…
```
customer_type
Transient          89613
Transient-Party    25124
Contract            4076
Group                577
Name: count, dtype: int64
```

Identifying inconsistent data in numerical values:

In [371…
```
bookings_data.adults.value_counts()
```

Out[371…
```
adults
2     89680
1     23027
3      6202
0       403
4        62
26        5
27        2
20        2
5         2
40        1
50        1
55        1
6         1
10        1
Name: count, dtype: int64
```

In [372…
```
bookings_data[['stays_in_week_nights','stays_in_weekend_nights']].value_c
```

Out[372…
```
stays_in_week_nights  stays_in_weekend_nights
2                     0                          17956
1                     0                          16451
3                     0                          11564
2                     1                           8979
5                     2                           8655
                                                  ...
26                    12                             1
17                    7                              1
25                    9                              1
18                    8                              1
50                    19                             1
Name: count, Length: 85, dtype: int64
```

Conclusion:

After identifying unique values, some inconsistencies are found such as:

- bookings with 0 adults
- bookings where there are no days to stay
- Undefined market segments
- Undefined meal types

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.'

In [425…

```
# Reservations with zero adults

bookings_data = bookings_data.drop(bookings_data[bookings_data['adults']
```

In [426…

```
# Examining rows with zeros in both 'stays_in_weekend_nights' and 'stays_

bookings_data = bookings_data.drop(bookings_data[(bookings_data['stays_in
```

In [427…

```
# After dropping all these columns an index reset is needed
bookings_data.reset_index(drop=True, inplace=True)
```

Handling Outliers

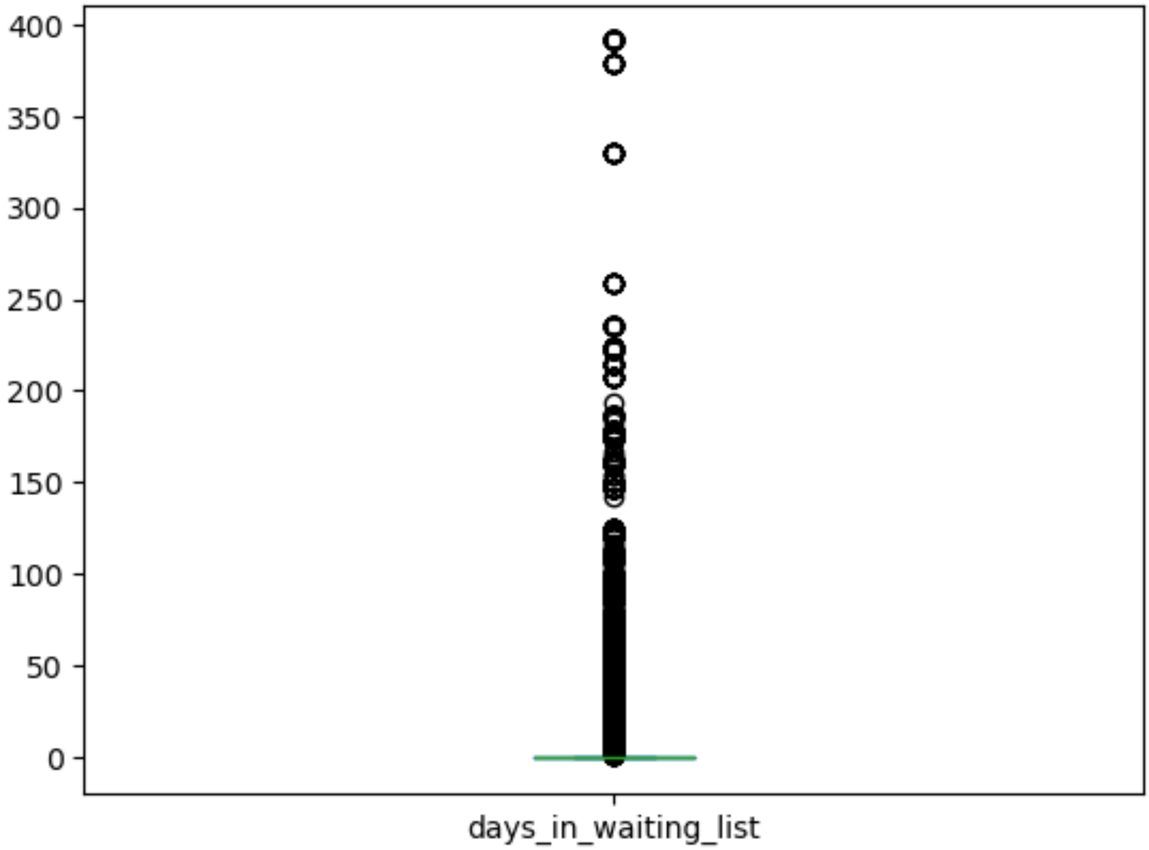
Days in Waiting List column

In [377…

```
bookings_data.plot(y=['days_in_waiting_list'],kind='box')
# Outliers are vital for enhancing classification accuracy and insights.
```

Out[377…

```
<Axes: >
```



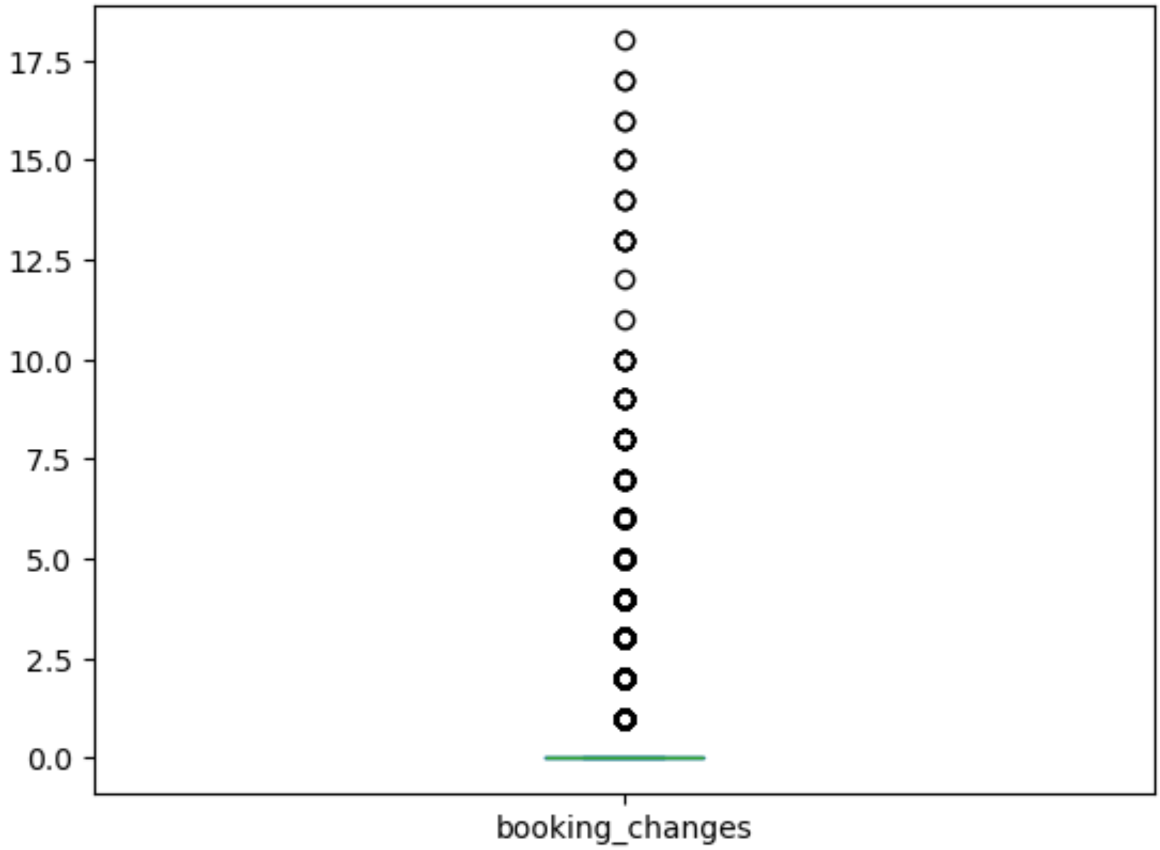
Booking Changes Column

In [378…

```
bookings_data.plot(y=['booking_changes'],kind='box')
# Outliers are vital for enhancing classification accuracy and insights.
```

Out[378…

```
<Axes: >
```



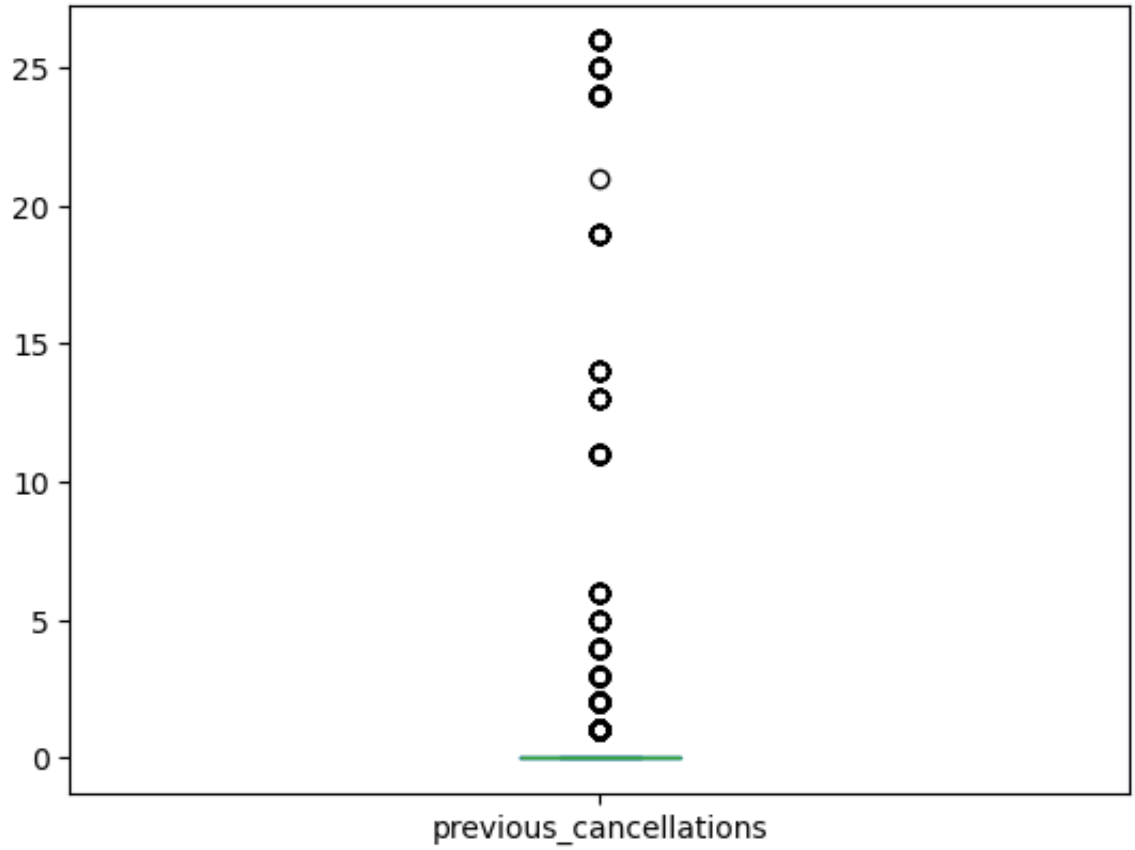
Previous Cancellations Column

In [379…

```
bookings_data.plot(y=['previous_cancellations'],kind='box')
# Outliers are vital for enhancing classification accuracy and insights.
```

Out[379…

```
<Axes: >
```



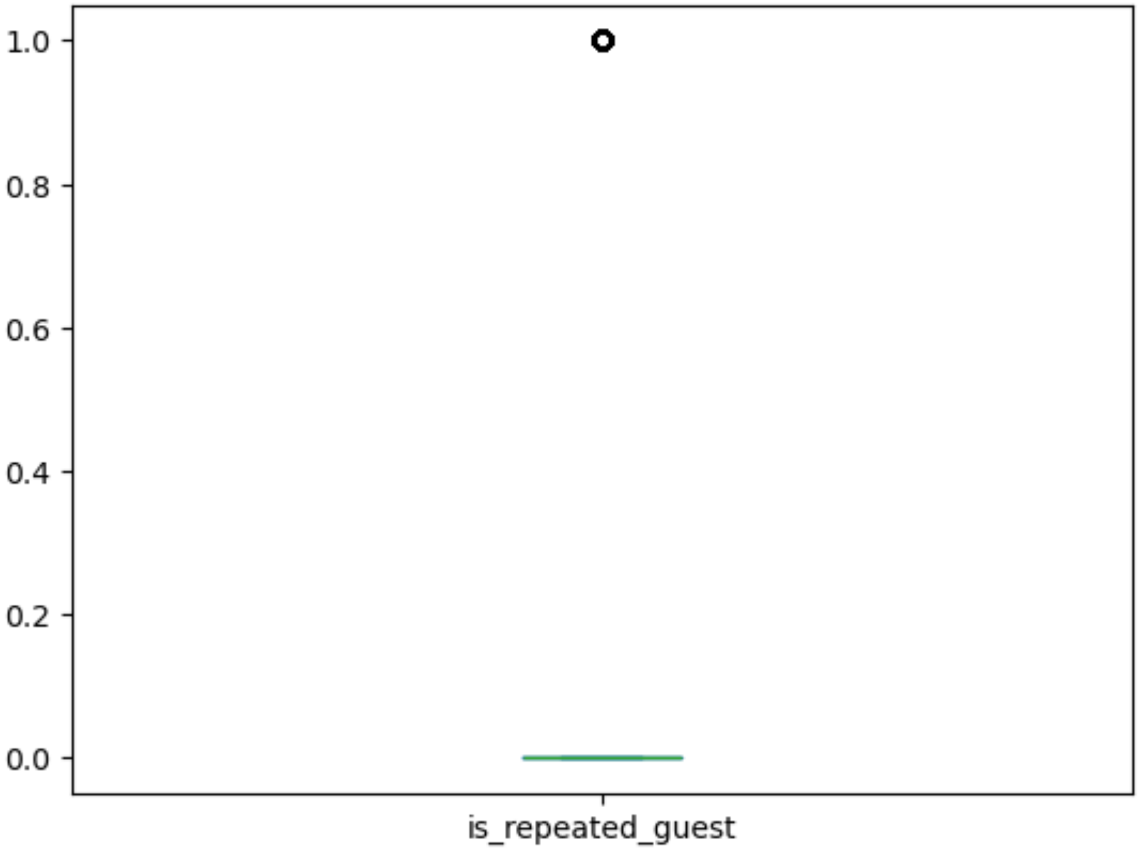
Is Repeated Guest Column

In [380…

```
bookings_data.plot(y=['is_repeated_guest'],kind='box')
# Outliers are vital for enhancing classification accuracy and insights.
```

Out[380…

```
<Axes: >
```



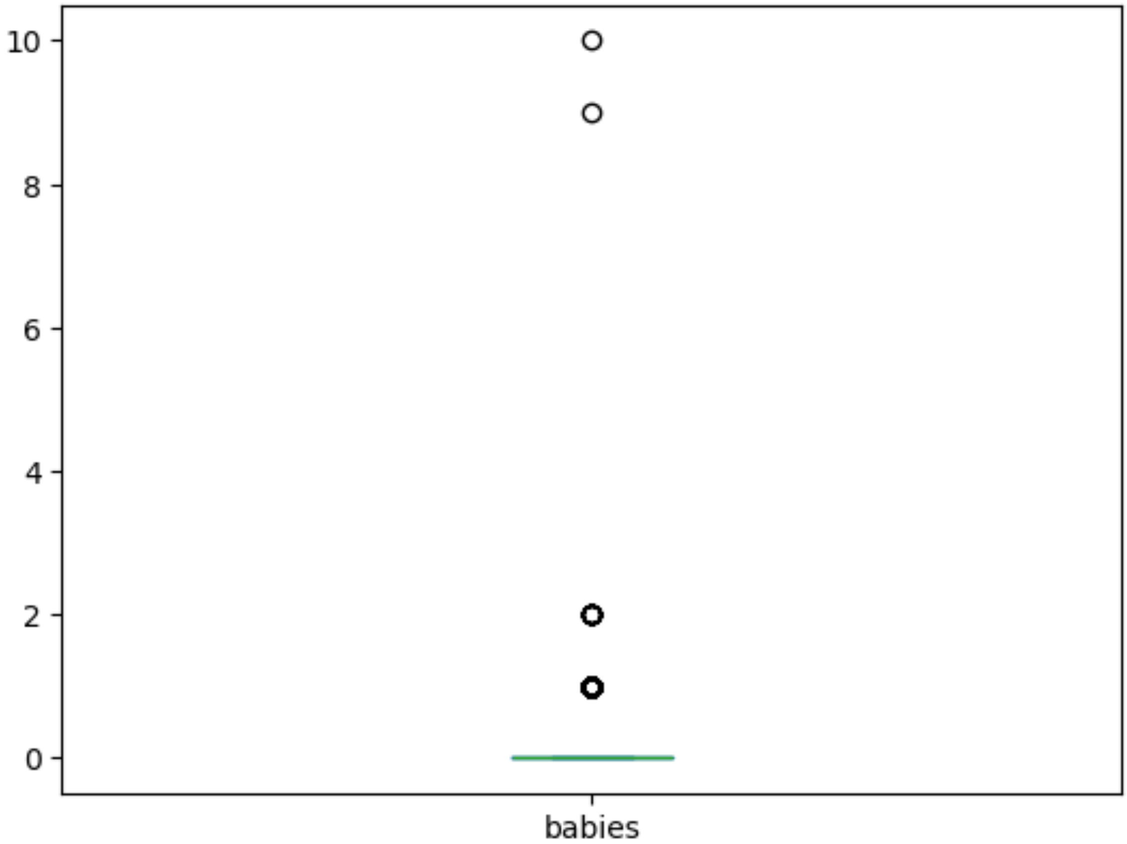
Babies Column

In [381…

```
bookings_data.plot(y=['babies'],kind='box')
# Outliers are vital for enhancing classification accuracy and insights.
```

Out[381…

```
<Axes: >
```



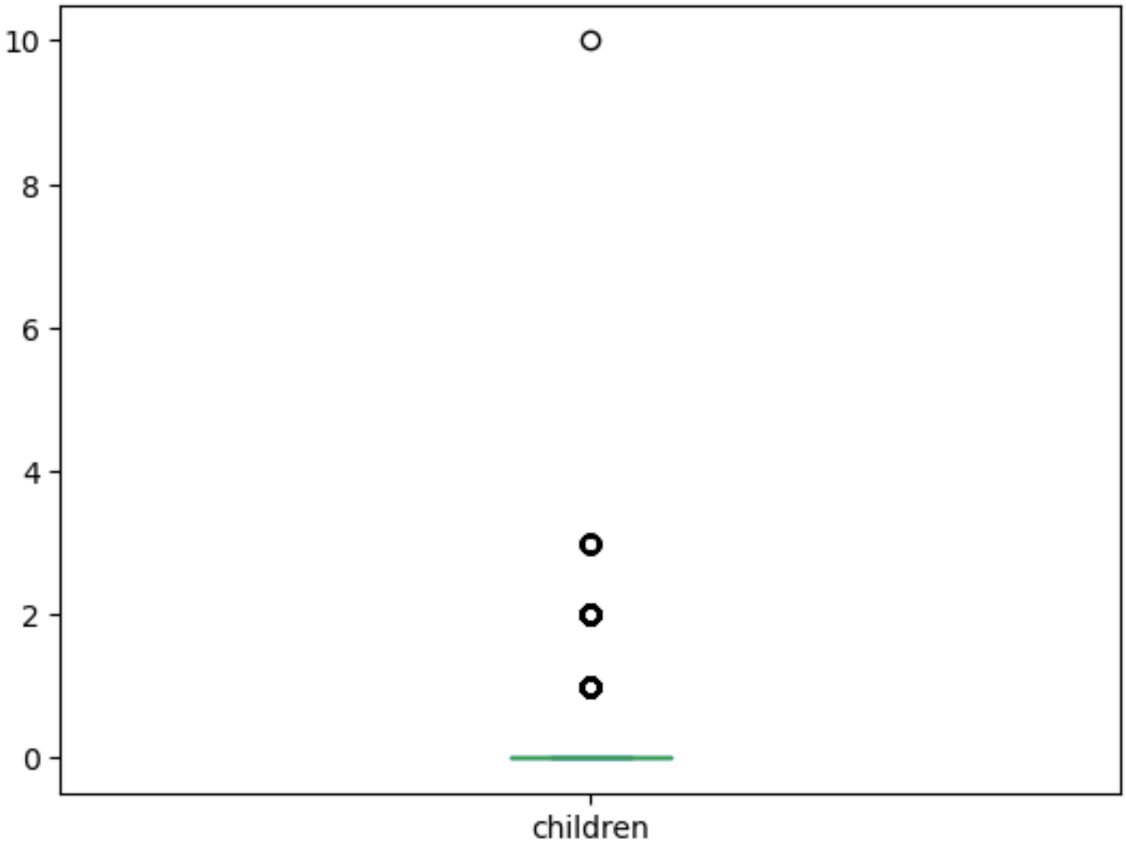
Children Column

In [382…

```
bookings_data.plot(y=['children'],kind='box')
# Outliers are vital for enhancing classification accuracy and insights.
```

Out[382…

```
<Axes: >
```

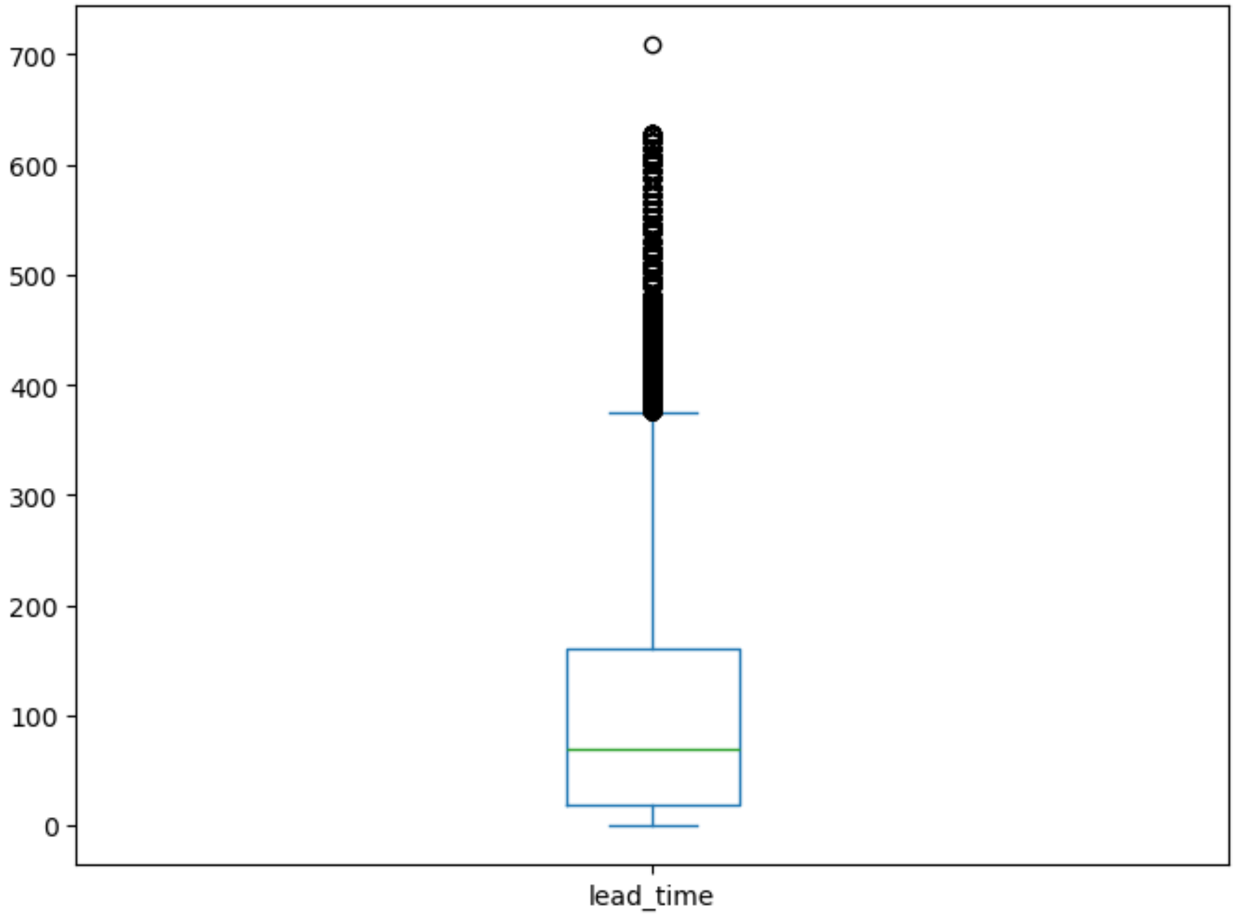


Lead Time Column

In [383…
```
bookings_data.plot(y=['lead_time'],kind='box',figsize=[8,6])
```

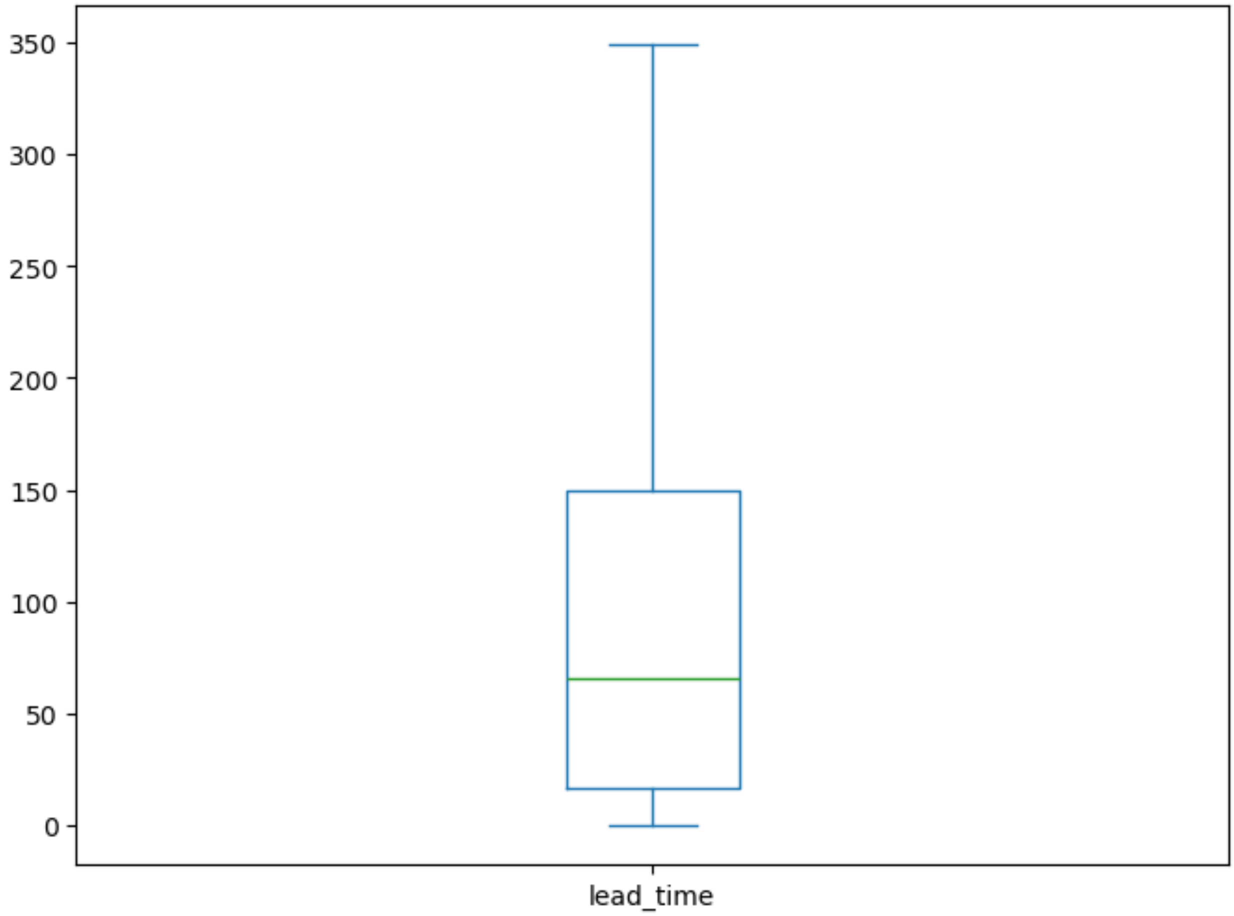
Out[383…

```
<Axes: >
```



In [384…
```
bookings_data = bookings_data.drop(bookings_data[bookings_data['lead_time
bookings_data.plot(y=['lead_time'],kind='box',figsize=[8,6])
```

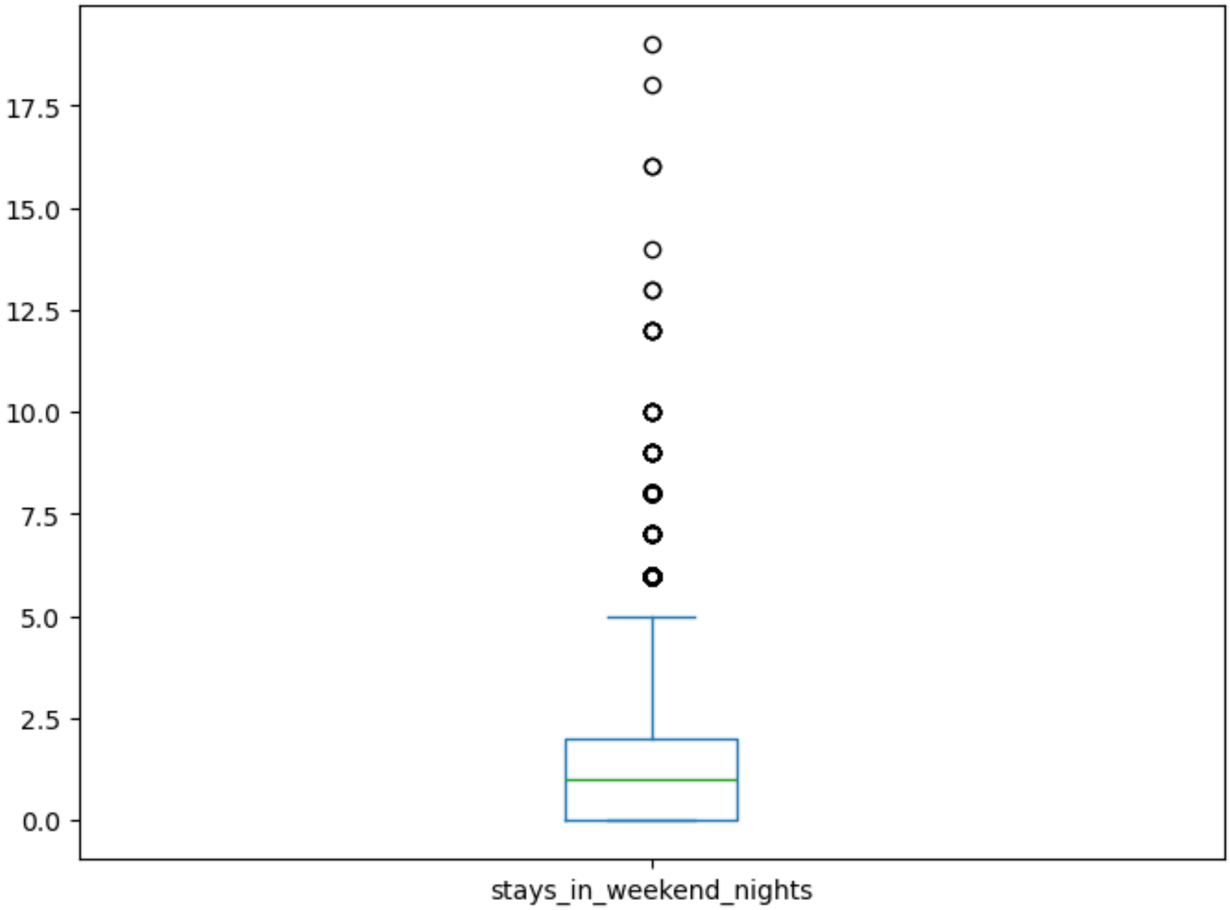
Out[384…
```
<Axes: >
```



Stays in Weekend Nights Column

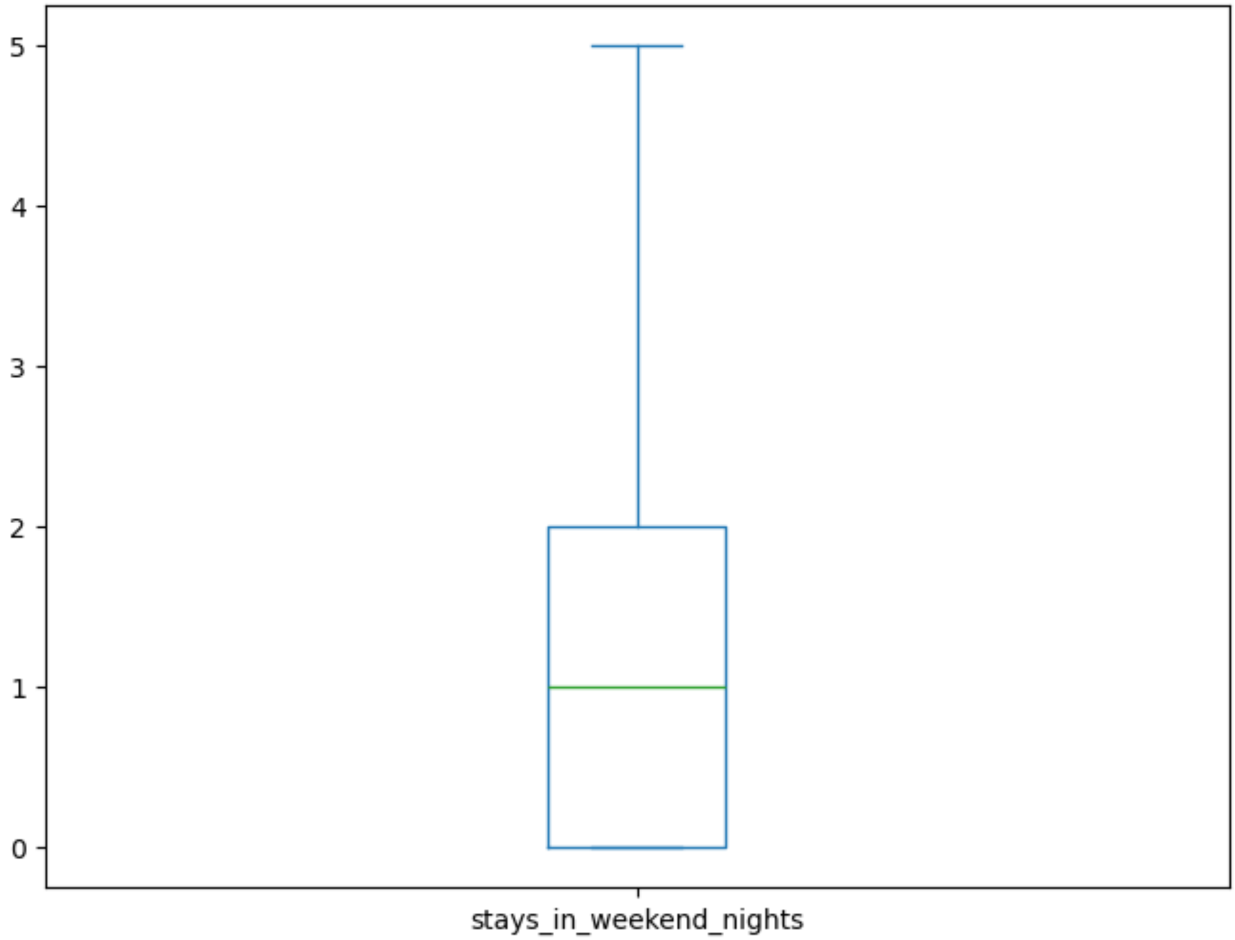
In [385…
```
bookings_data.plot(y=['stays_in_weekend_nights'],kind='box',figsize=[8,6]
```

Out[385…
```
<Axes: >
```



In [386…
```
bookings_data = bookings_data.drop(bookings_data[bookings_data['stays_in_
bookings_data.plot(y=['stays_in_weekend_nights'],kind='box',figsize=[8,6]
```

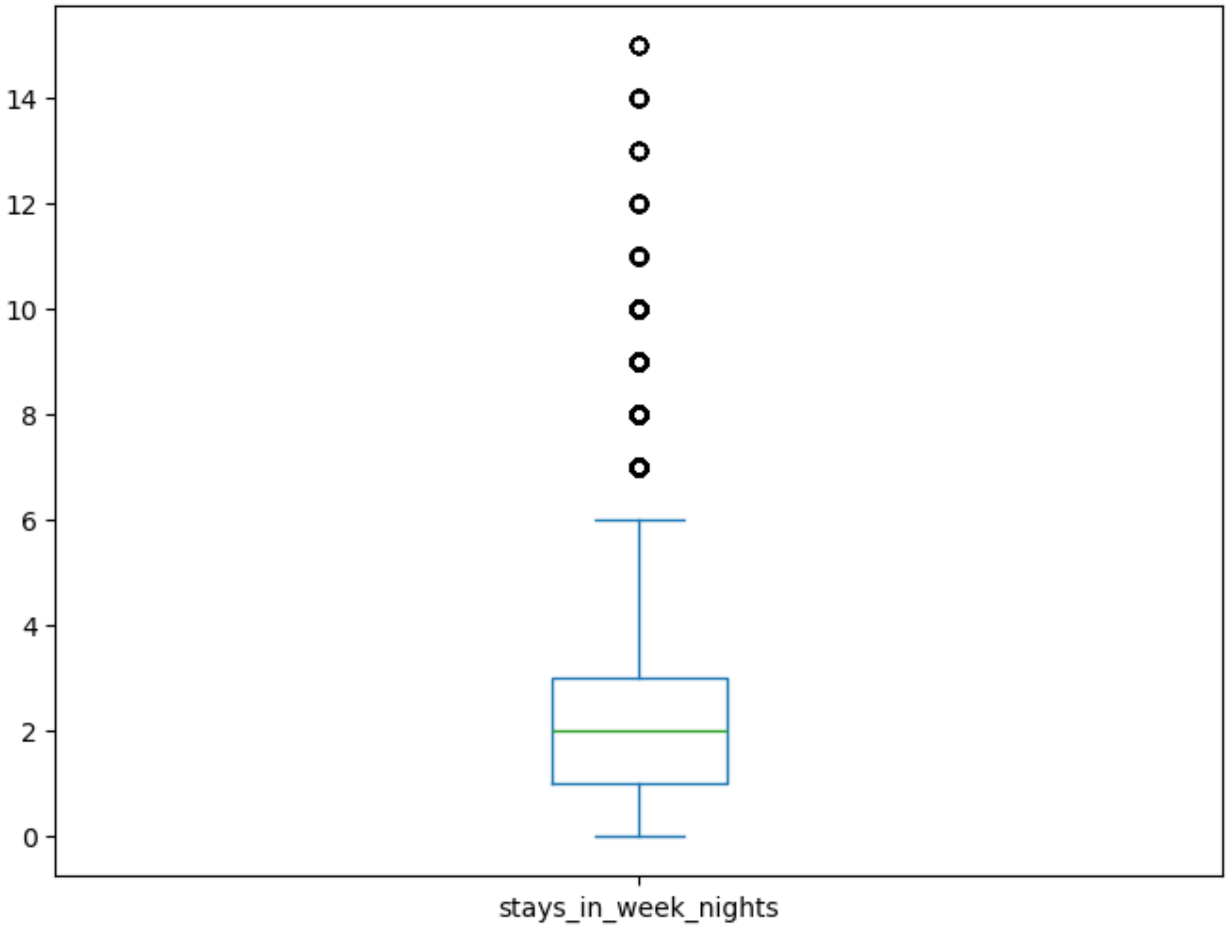
Out[386…
```
<Axes: >
```



Stays in Week Nights Column

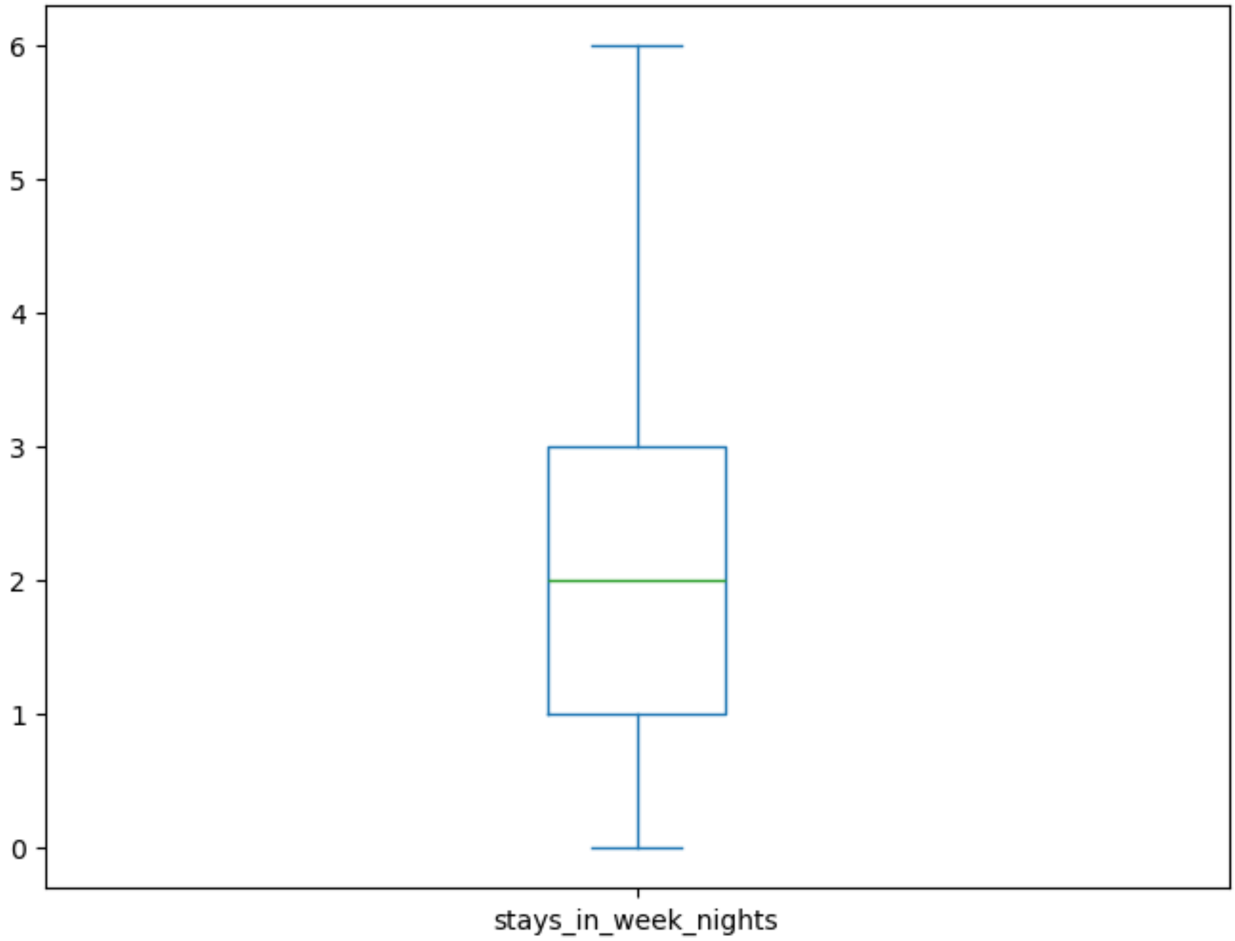
In [387…
```
bookings_data.plot(y=['stays_in_week_nights'],kind='box',figsize=[8,6])
```

Out[387…
```
<Axes: >
```



In [388…
```
bookings_data = bookings_data.drop(bookings_data[bookings_data['stays_in_
bookings_data.plot(y=['stays_in_week_nights'],kind='box',figsize=[8,6])
```

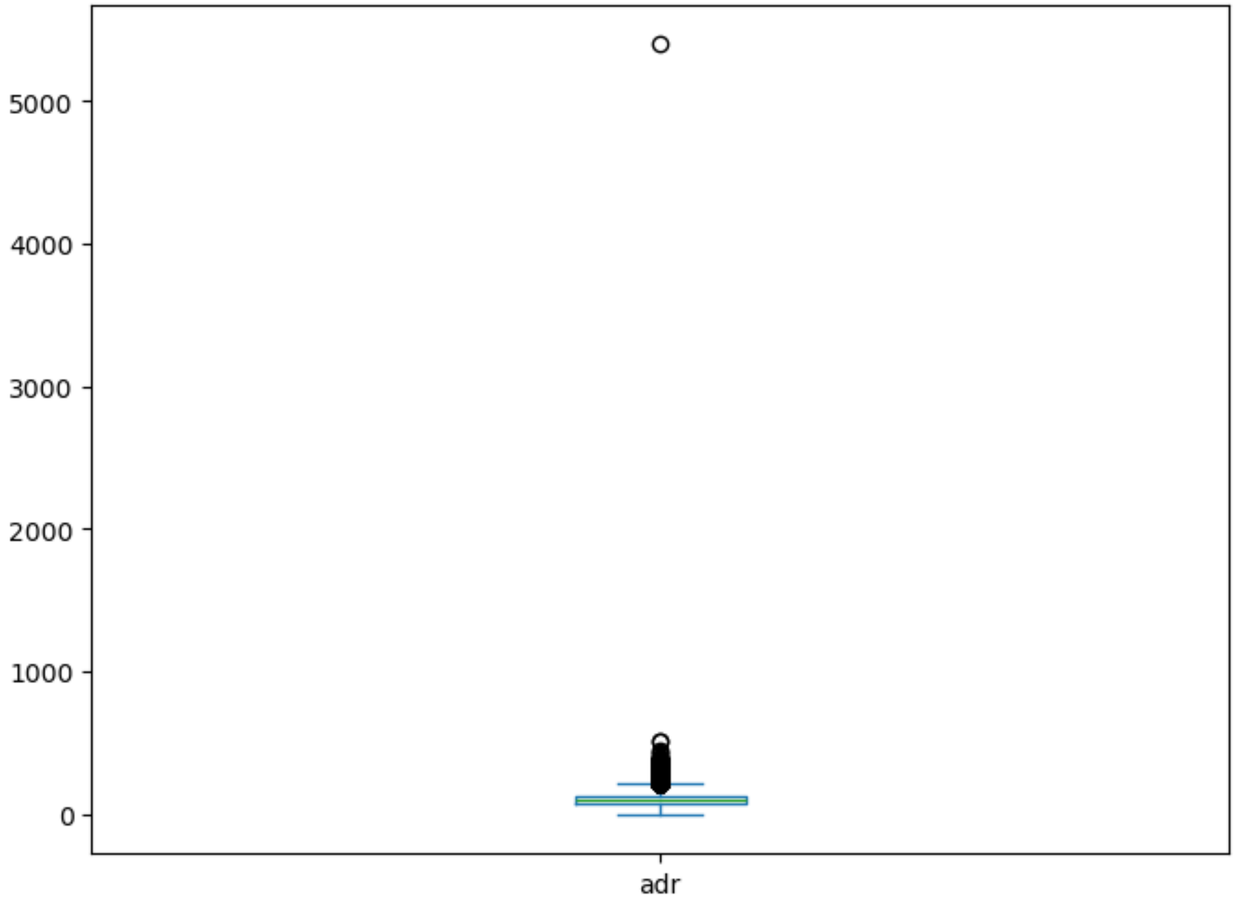
Out[388…
```
<Axes: >
```



ADR Column

In [389…
```
bookings_data.plot(y=['adr'],kind='box',figsize=[8,6])
```

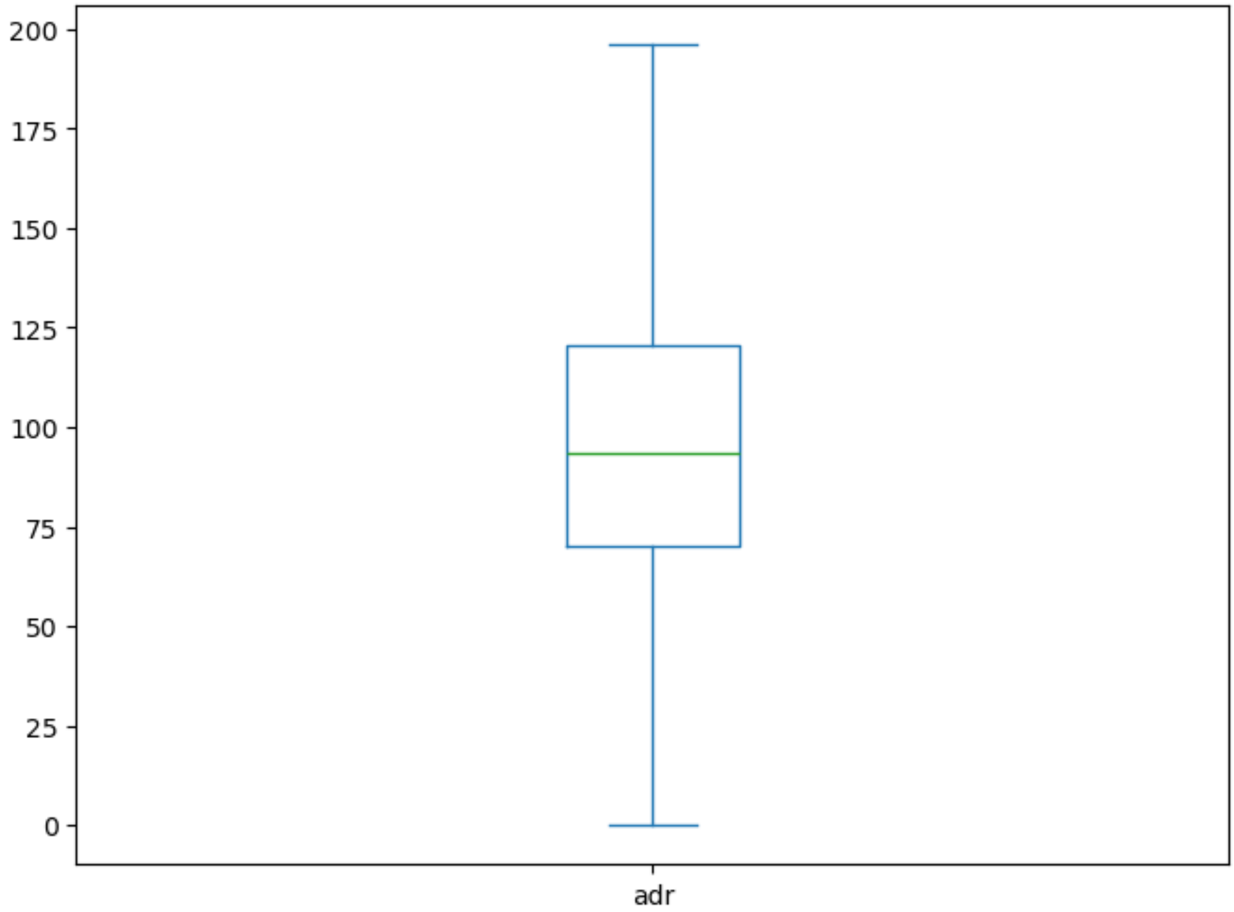
Out[389…
```
<Axes: >
```



In [390…
```
bookings_data = bookings_data.drop(bookings_data[(bookings_data['adr'] >

bookings_data.plot(y=['adr'],kind='box',figsize=[8,6])
```

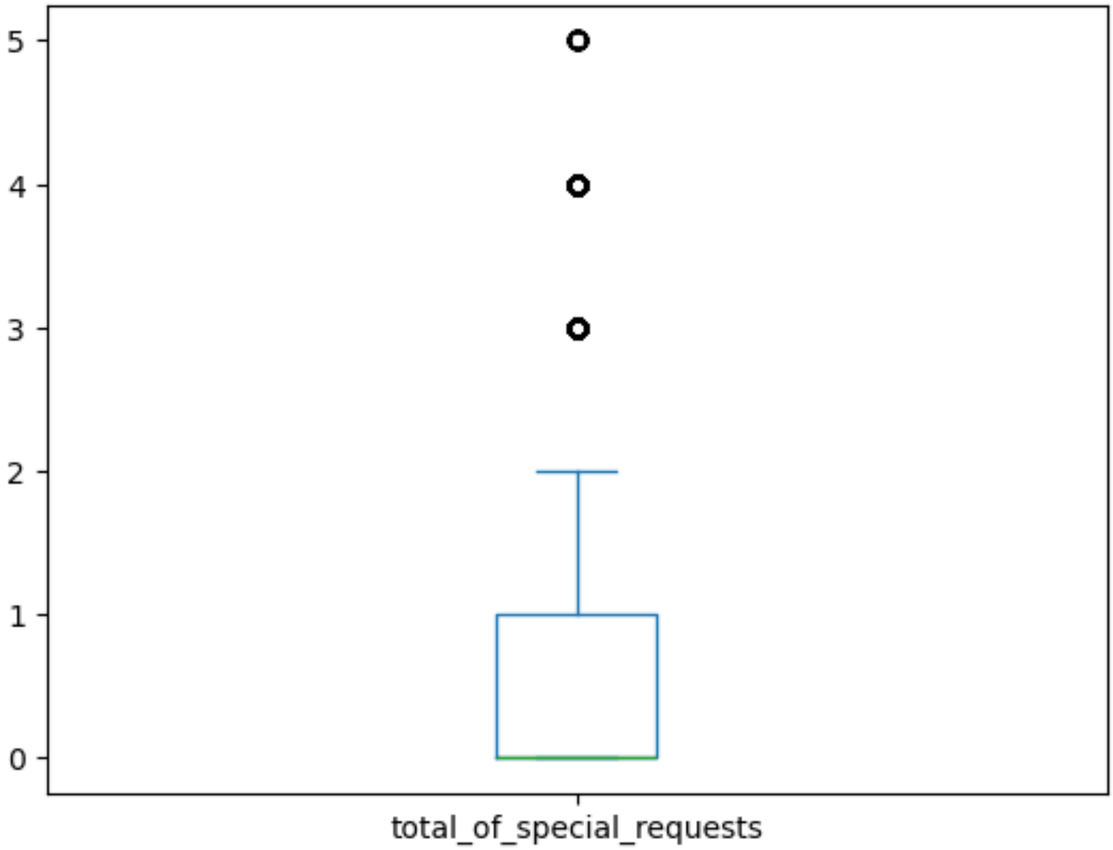
Out[390…
```
<Axes: >
```



Total Special Requests Column

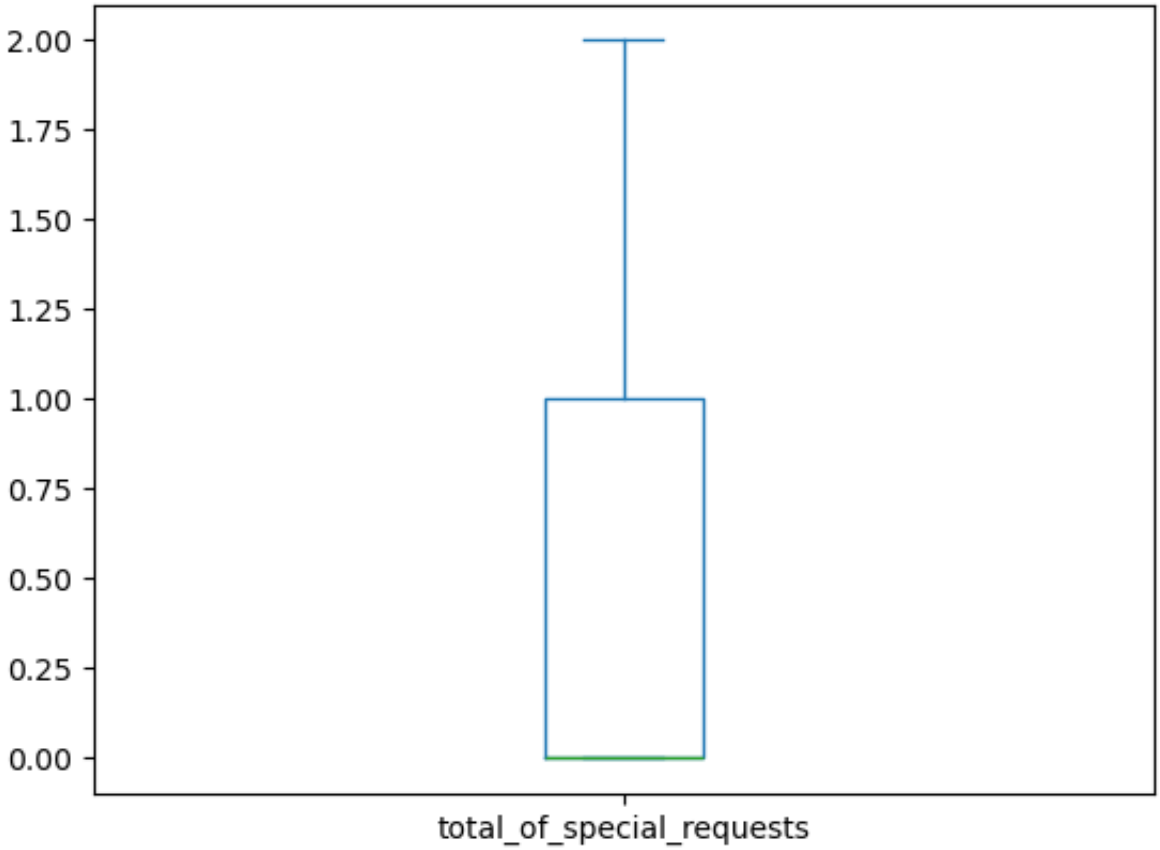
In [391…
```
bookings_data.plot(y=['total_of_special_requests'],kind='box')
```

Out[391…
```
<Axes: >
```



In [392…
```
bookings_data = bookings_data.drop(bookings_data[bookings_data['total_of_
bookings_data.plot(y=['total_of_special_requests'],kind='box')
```

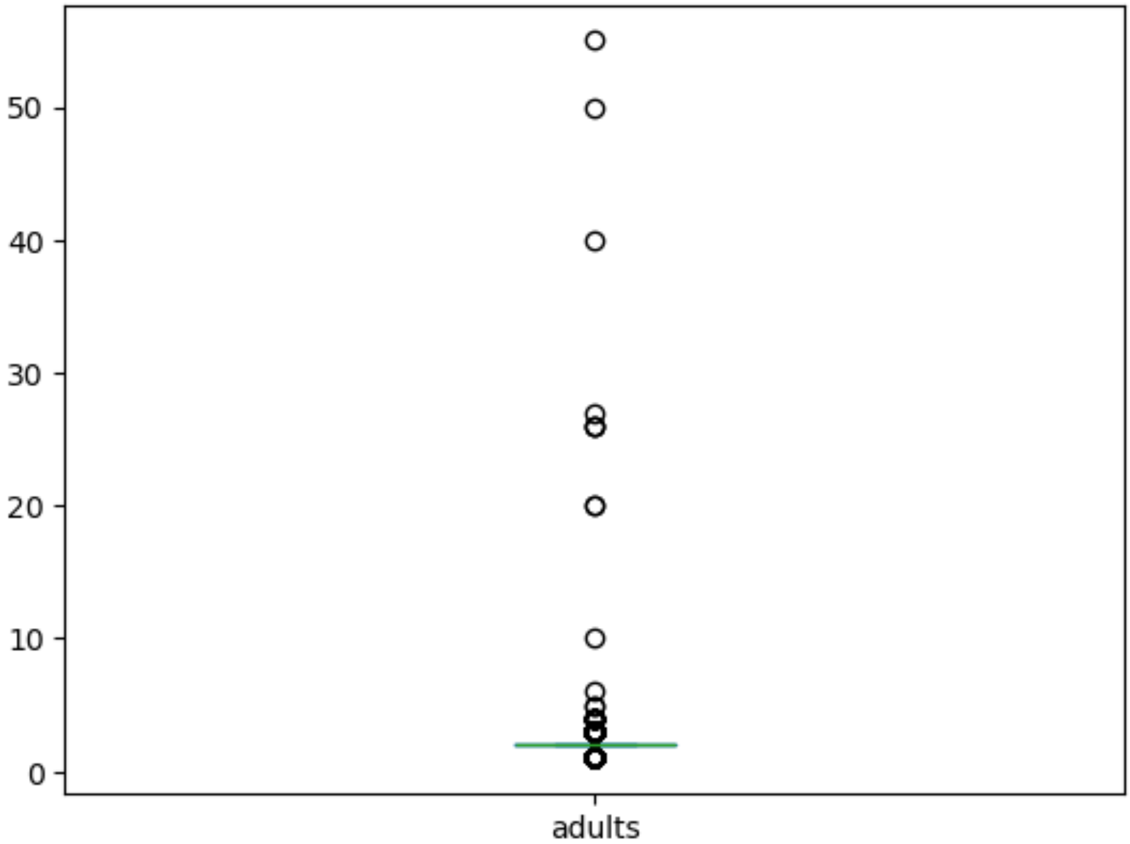
Out[392…
```
<Axes: >
```



Adults Column

In [393…
```
bookings_data.plot(y=['adults'],kind='box')
# Outliers are vital for enhancing classification accuracy and insights.
```

Out[393… <Axes: >



1.3 Column data type conversion (5%)

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

The "Children" column should be of integer data type, as it represents the count of children and should not contain floating-point values.

In [394…
```
bookings_data['children'] = bookings_data['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.

Splitting Data for hotels

In [395…

```
hotel_name = bookings_data.hotel.unique()
```

In [396…

```
city_hotel_data = bookings_data.drop(bookings_data[bookings_data['hotel']

resort_hotel_data = bookings_data.drop(bookings_data[bookings_data['hotel

# This code splits each hotel data into two different datasets so we can
```

In [397…

```
# Getting the is_cancelled column into a variable
resort_is_canceled = resort_hotel_data.is_canceled.value_counts()
```

In [398…

```
# Getting the is_cancelled column into a variable
city_is_canceled = city_hotel_data.is_canceled.value_counts()
```

Solution

To analyze the cancellation percentages for City and Resort hotels, I have compiled the data to highlight the comparative cancellation rates for each type of hotel.

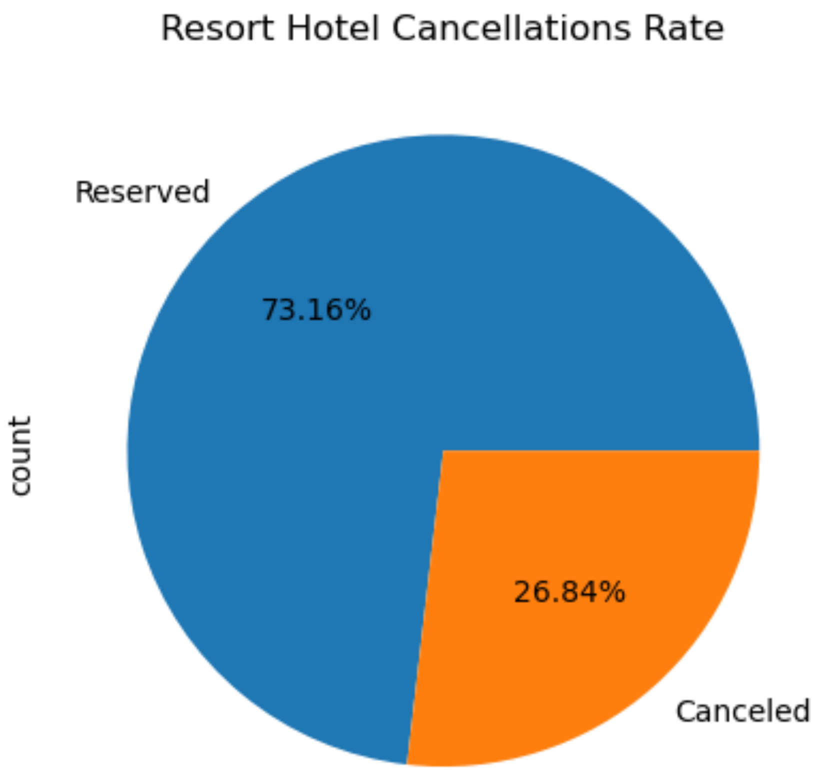
In [399…

```
# Resort Hotel Pie

resort_is_canceled.plot(kind='pie',labels={'Reserved','Canceled'},title='
```

Out[399…

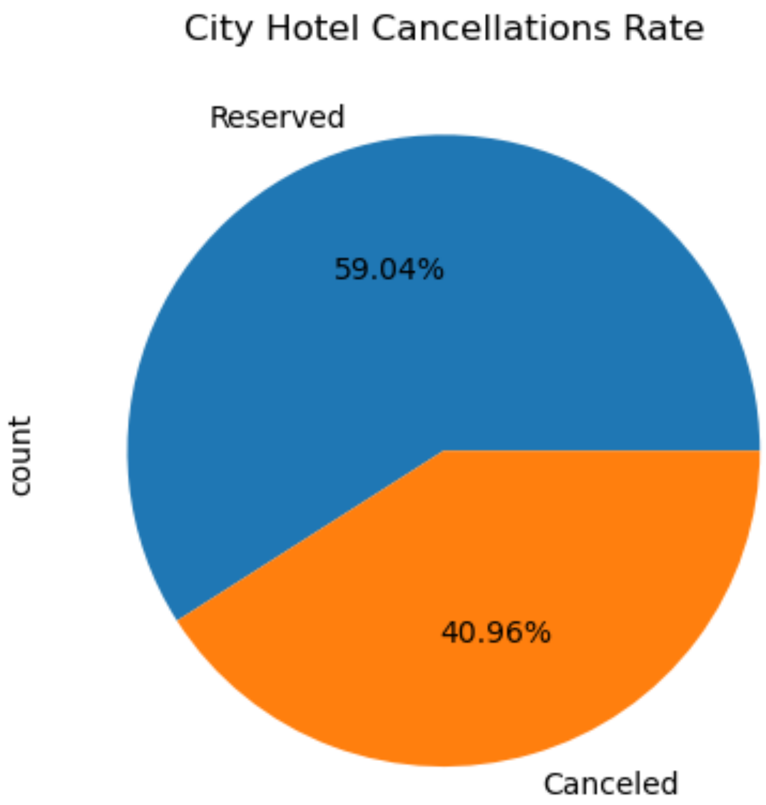
```
<Axes: title={'center': 'Resort Hotel Cancellations Rate'}, ylabel='coun
t'>
```



In [400…
```
# City Hotel Pie

city_is_canceled.plot(kind='pie',labels={'Reserved','Canceled'},title='Ci
```

Out[400…
```
<Axes: title={'center': 'City Hotel Cancellations Rate'}, ylabel='count'
>
```



Conclusion

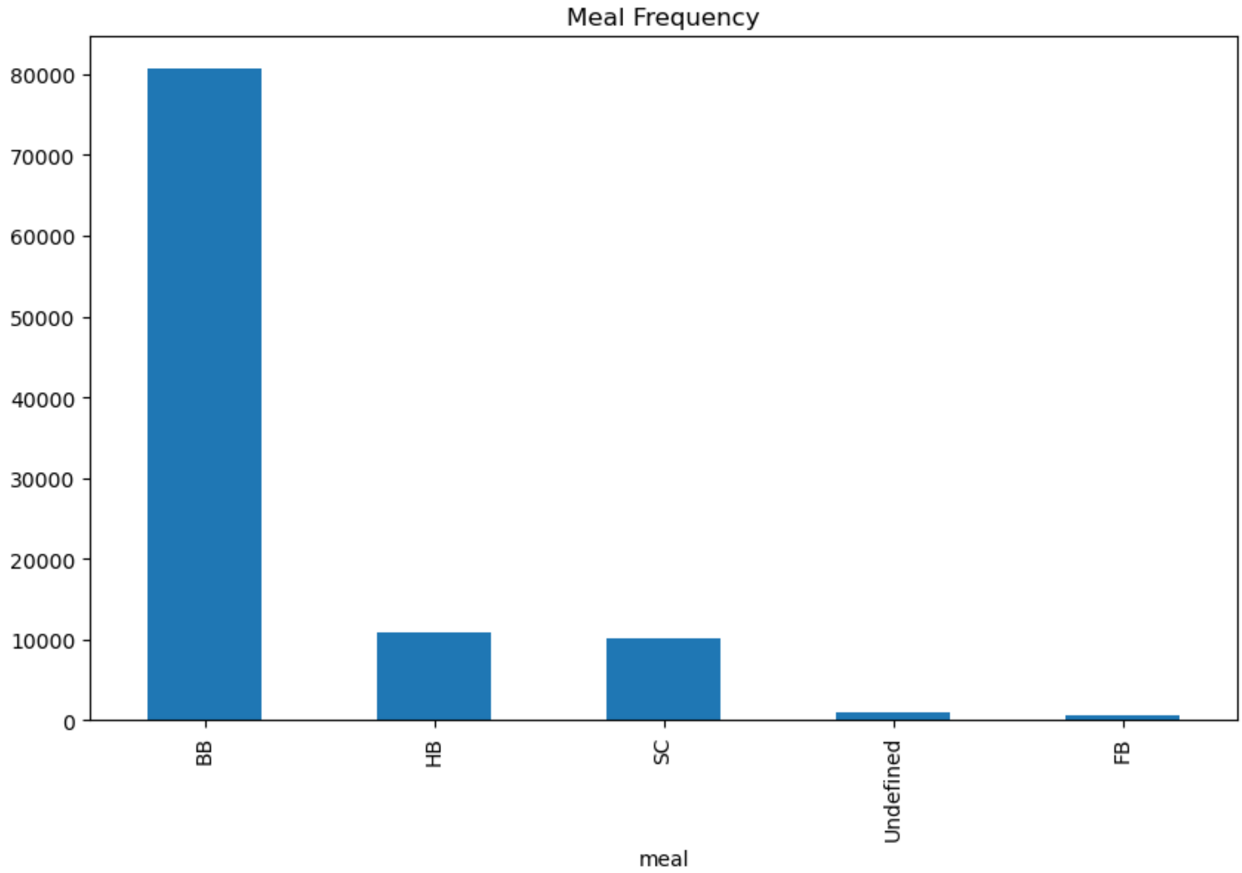
City Hotels exhibit a cancellation rate of 40.96%, while Resort Hotels show a cancellation rate of 26.84%. This comparison indicates that City Hotels experience a higher cancellation rate than Resort Hotels, which may suggest differing traveler behaviors or market conditions.

2.2. Identifying the most frequently ordered meal types.

In [401…
```
# To solve this task we simply need to show the value counts of the meal

bookings_data.meal.value_counts().plot(kind='bar',title='Meal Frequency',
```

Out[401…
```
<Axes: title={'center': 'Meal Frequency'}, xlabel='meal'>
```



Solution

As observable in the bar chart the Bread and Breakfast is ordered over 80000 times making it the most popular and frequent type of meal.

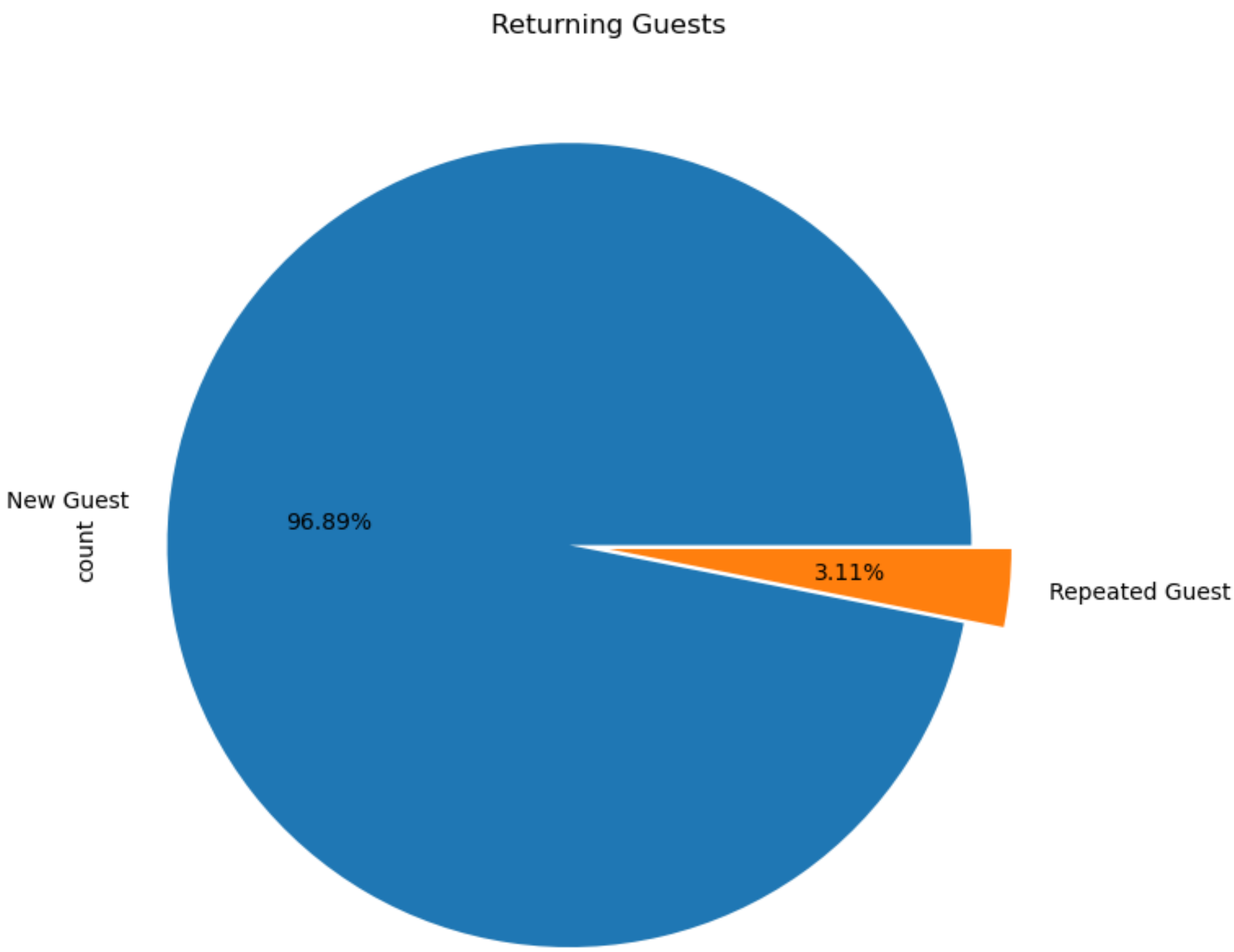
2.3. Determining the number of returning guests.

In [402…

```
explode = (0.1,0)
bookings_data.is_repeated_guest.value_counts().plot(kind='pie',labels={'N
```

Out[402…

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



In [403…

```
repeat_guests = bookings_data.is_repeated_guest.value_counts()
repeat_guests
```

Out[403…

```
is_repeated_guest
0    100204
1      3218
Name: count, dtype: int64
```

Solution

The total amount of repeated guests is 3218

2.4. Discovering the most booked room types.

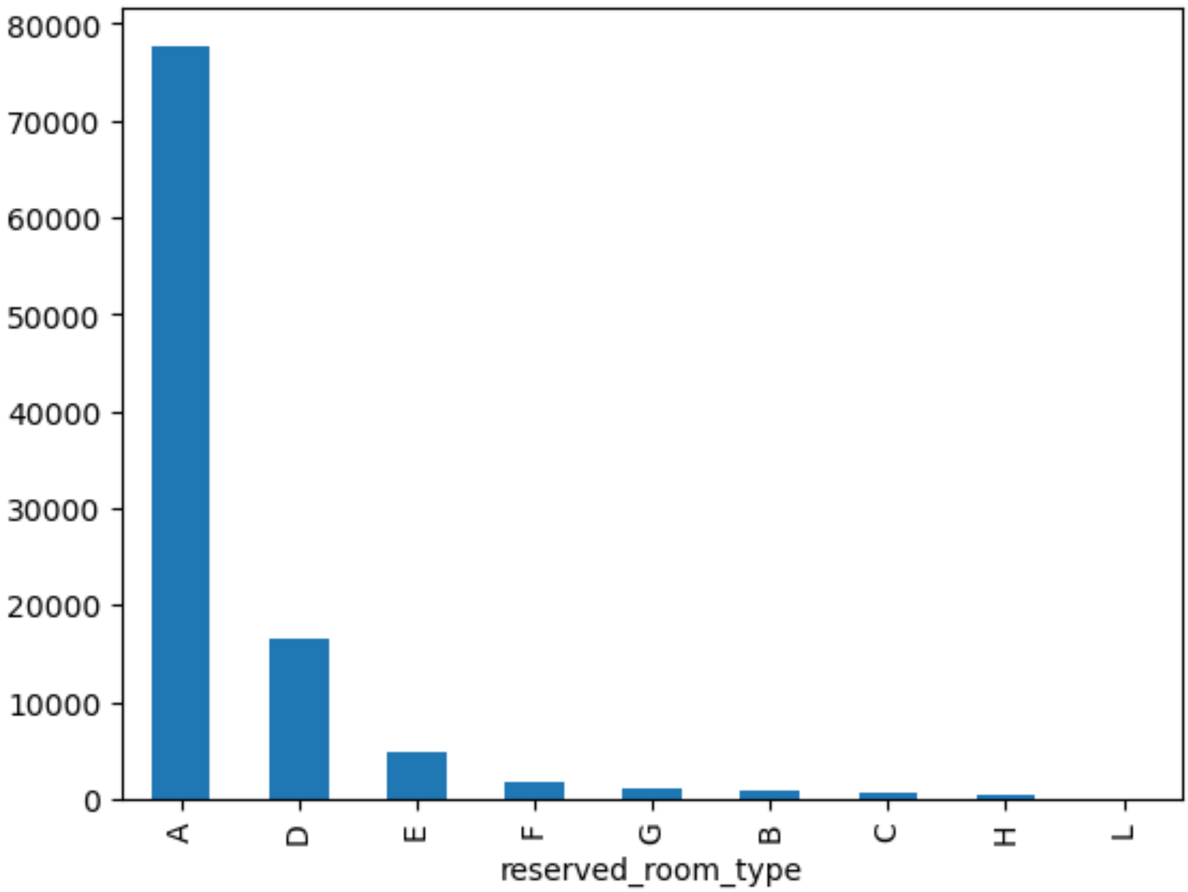
In [404…

```
room_type_chart = bookings_data.reserved_room_type.value_counts()
room_type_chart
```

Out[404…
```
reserved_room_type
A    77693
D    16410
E     4757
F     1742
G     1104
B      836
C      546
H      329
L        5
Name: count, dtype: int64
```

In [405…
```
room_type_chart.plot(kind='bar')
```

Out[405…
```
<Axes: xlabel='reserved_room_type'>
```



Solution

The analysis indicates that Room A is the most frequently booked accommodation option.

2.5. Exploring correlations between room types and cancellations.

In [406…

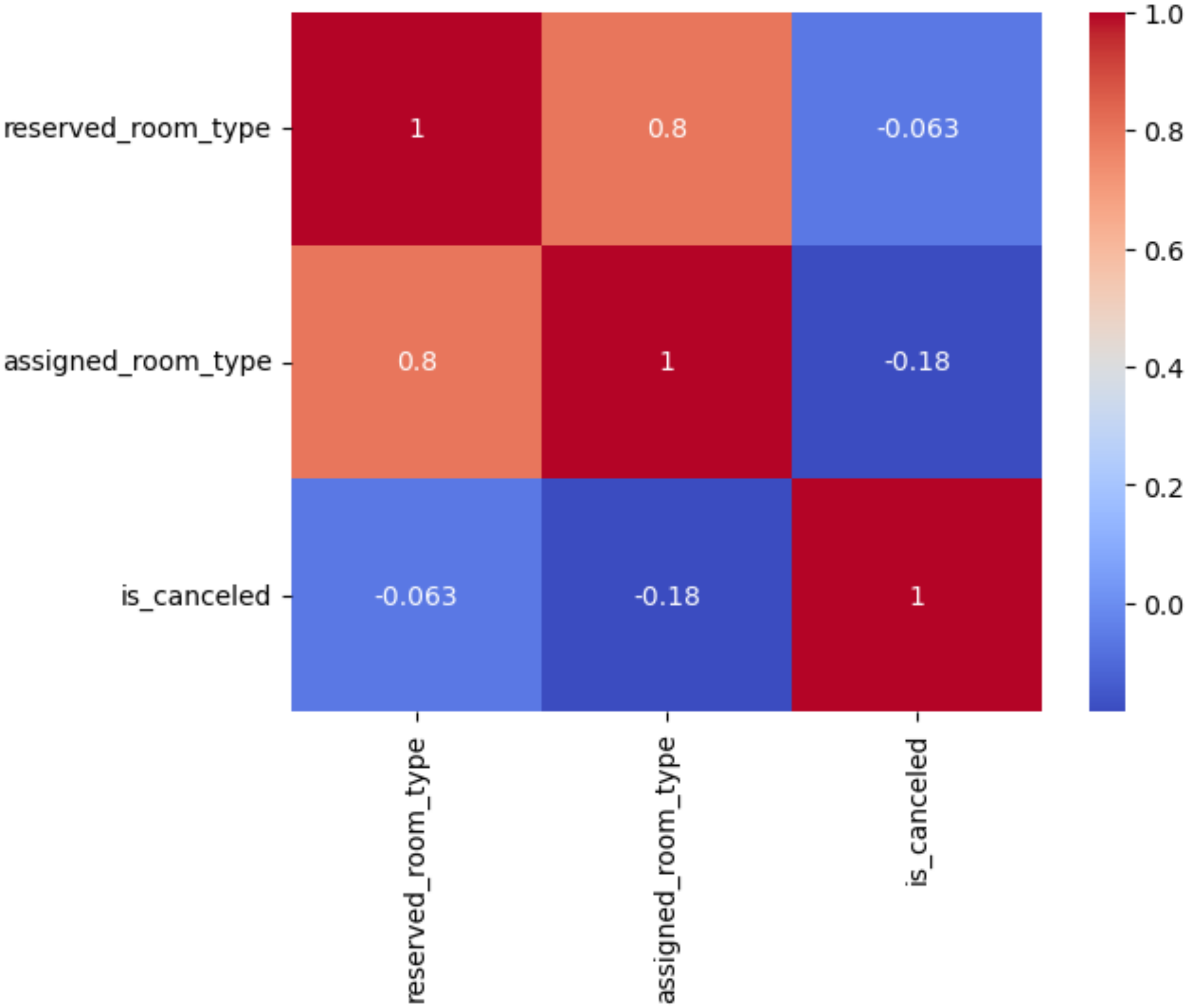
```
bookings_data['reserved_room_type'] = bookings_data['reserved_room_type']
bookings_data['assigned_room_type'] = bookings_data['assigned_room_type']


room_cancellations = bookings_data[['reserved_room_type','assigned_room_t

corr = room_cancellations.corr().round(3)

sb.heatmap(corr,cmap="coolwarm",annot=True)
```

Out[406… `<Axes: >`



Solution

1. Correlation between Assigned Room Type and Reserved Room Type (80%): The strong positive correlation of 80% indicates that there is a significant alignment between the room types that guests are assigned and the types they originally reserved. This suggests that the hotel's allocation practices are largely effective in matching reservations to actual assignments. Such a high correlation may imply efficient management of room inventory and customer expectations, leading to higher satisfaction rates.

2. Inverse Correlation between Assigned Room Type and Cancellation Status (20%): The 20% inverse correlation between the assigned room type and the cancellation status implies that as the likelihood of a specific room type being assigned increases, the probability of cancellations decreases. Although this correlation is weaker, it still suggests that customers who receive their preferred room types are less likely to cancel their bookings. This insight could inform strategies to minimize cancellations by ensuring that customers are assigned the room types they originally selected.

Conclusion

In summary, these correlations highlight the importance of aligning assigned and reserved room types to enhance guest satisfaction and reduce cancellations. Further investigation could explore the underlying factors contributing to these relationships and inform strategies for improving booking management.

3. Feature Engineering (20%)

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

Hint:

- Binning
- Encoding
- Scaling
- Feature selection

3.1. Binning

Binning explaination:

Binning is used only on continuous numerical data. Binning simplifies the data and can reveal trends by transforming a large range of continuous values into smaller, more manageable categories.

I used the Sturges' formula for the number bins of each column:

Number of bins = [log2(n) + 1]

Continuous Numerical Data columns:

- Lead time
- ADR (Average Daily Rate)

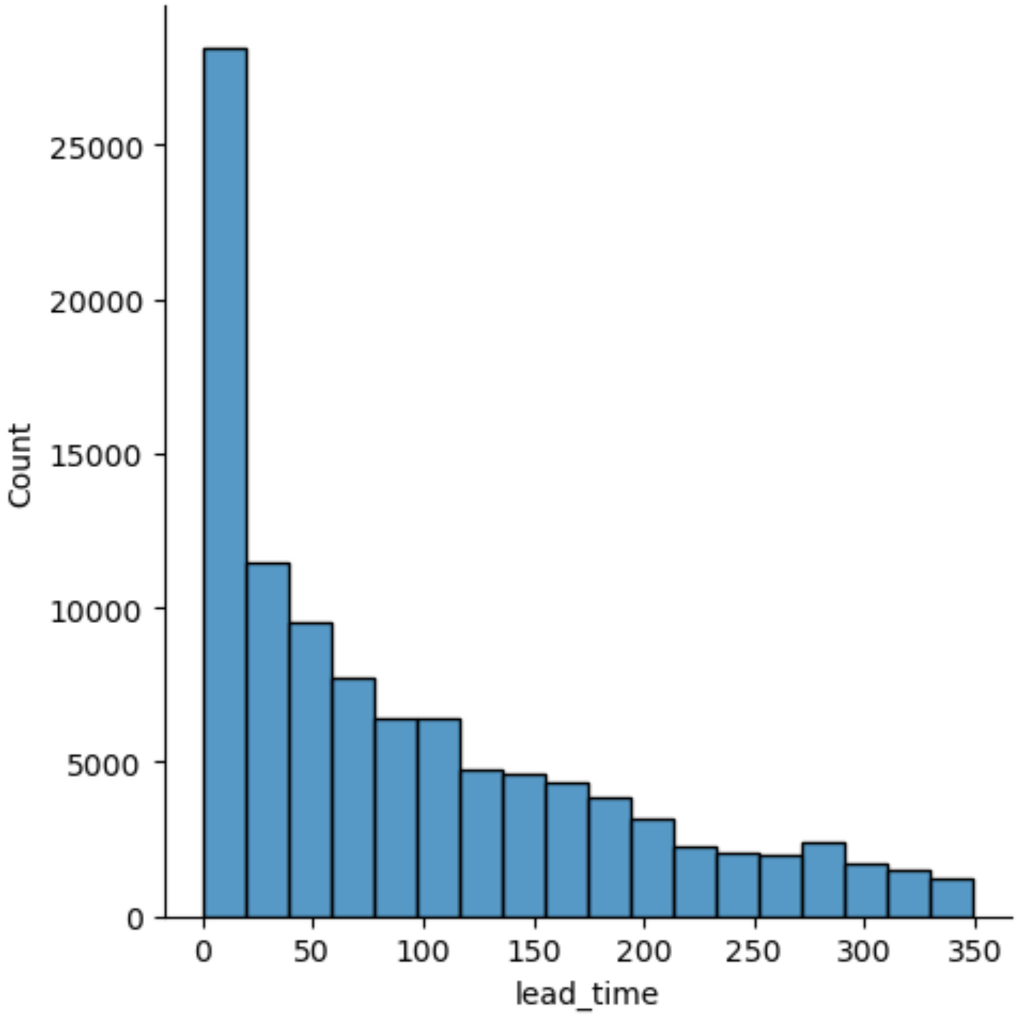
In [407…
```
# Calculating the number of data collected:
collected_data_number = bookings_data.shape[0]
```

In [408…
```
# Applying Sturges formula to get the appropiate number of bins
bins = (math.log2(collected_data_number) + 1).__round__()
bins
```

Out[408… 18

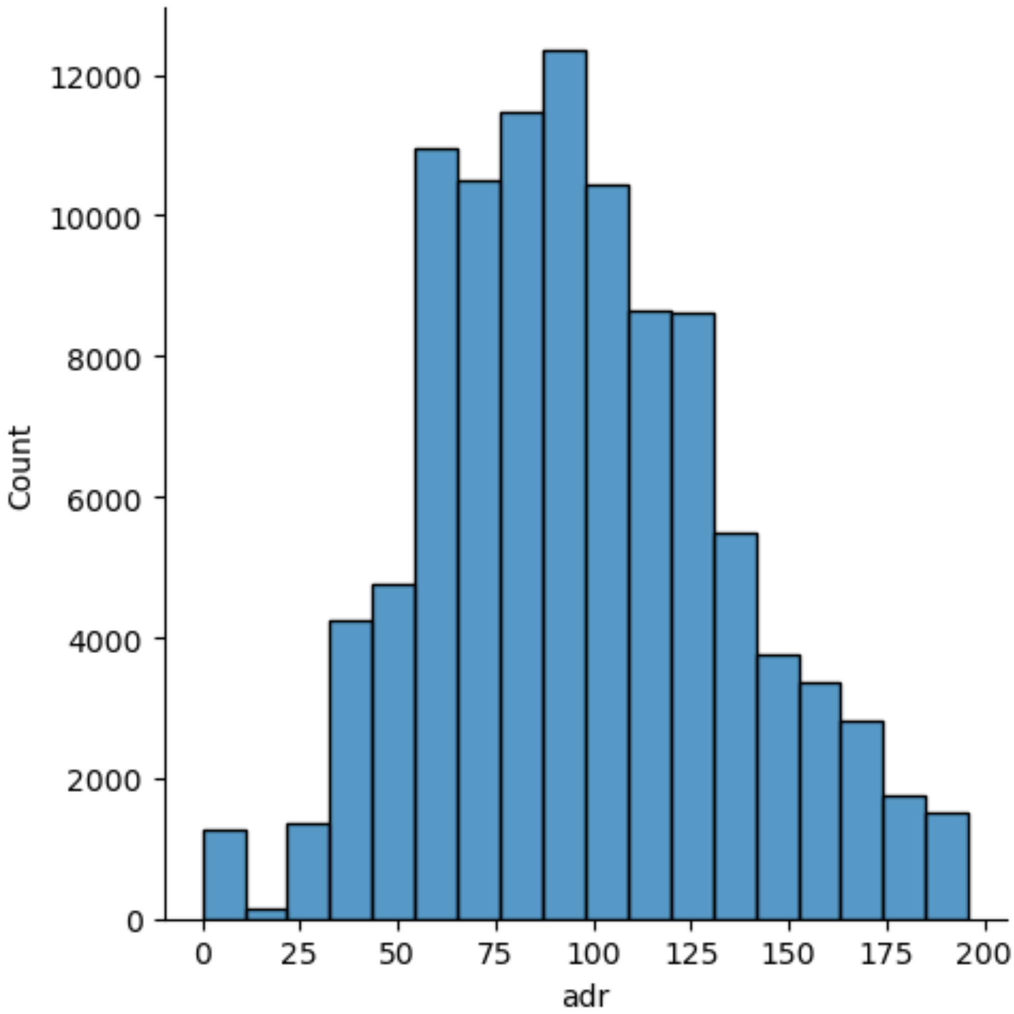
In [409…
```
# Lead time column
sb.displot(data=bookings_data, x='lead_time',bins = bins)
```

Out[409… <seaborn.axisgrid.FacetGrid at 0x337265760>



In [432…
```
# Adr column
sb.displot(data=bookings_data, x='adr', bins = bins)
```

Out[432… <seaborn.axisgrid.FacetGrid at 0x338681760>



3.2. Encoding

Hotel Column Encoding

In [411…

```
# Using pandas cat.codes (Binary Column)
bookings_data['hotel'] = bookings_data['hotel'].astype('category').cat.co
```

Meal Column Encoding

In [412…

```
# Index Clean before encoding:
bookings_data.reset_index(drop=True, inplace=True)
```

In [413…

```
# Using one hot encoding scikit learn library
from sklearn.preprocessing import OneHotEncoder

# Initialize OneHotEncoder from sklearn
ohe = OneHotEncoder(sparse_output=False)

ohe_coded = ohe.fit_transform(bookings_data[['meal']])
```

```
# Convert the result into a DataFrame with proper column names
one_hot_df = pd.DataFrame(ohe_coded,columns=ohe.get_feature_names_out(['m

# Drop the original 'meal' column
bookings_data = bookings_data.drop('meal', axis=1)

# Join the new one-hot encoded df back to the original
bookings_data = bookings_data.join(one_hot_df)
```

Market Segment Column

In [414…

```
# Apply OneHotEncoder to the 'distribution_channel' column
ohe_coded = ohe.fit_transform(bookings_data[['market_segment']])

# Convert the result into a DataFrame with proper column names
one_hot_df = pd.DataFrame(ohe_coded,columns=ohe.get_feature_names_out(['m

# Drop the original 'distribution_channel' column
bookings_data = bookings_data.drop('market_segment', axis=1)

# Join the new one-hot encoded df back to the original
bookings_data = bookings_data.join(one_hot_df)
```

Deposit Type Column

In [415…

```
deposit_type = bookings_data.deposit_type.unique()

# Using the manual replacing as its considerably small amount of unique v
bookings_data['deposit_type'] = bookings_data['deposit_type'].replace({de

bookings_data.deposit_type.value_counts()
```

```
/var/folders/yl/hc0wg3cd5577fyq8rq7spkx00000gn/T/ipykernel_54172/39146723
0.py:4: FutureWarning: Downcasting behavior in `replace` is deprecated and
will be removed in a future version. To retain the old behavior, explicitl
y call `result.infer_objects(copy=False)`. To opt-in to the future behavio
r, set `pd.set_option('future.no_silent_downcasting', True)`
  bookings_data['deposit_type'] = bookings_data['deposit_type'].replace({d
eposit_type[0]: 0, deposit_type[1]: 1, deposit_type[2]: 2})
```

Out[415…

```
deposit_type
0    91086
2    12186
1      150
Name: count, dtype: int64
```

Customer Type Column

In [416…

```
# Categorical Data encoding using scikitlearn one hot encoding:

# Apply OneHotEncoder to the 'customer_type' column
```

```
ohe_coded = ohe.fit_transform(bookings_data[['customer_type']])

# Convert the result into a DataFrame with proper column names
one_hot_df = pd.DataFrame(ohe_coded,columns=ohe.get_feature_names_out(['c

# Drop the original 'customer_type' column
bookings_data = bookings_data.drop('customer_type', axis=1)

# Join the new one-hot encoded df back to the original
bookings_data = bookings_data.join(one_hot_df)
```

Final Result

In [417…

```
# Data after encoding
bookings_data.head(10)
```

Out[417…

	hotel	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights
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
5	1	0	9	0	2
6	1	1	85	0	3
7	1	1	75	0	3
8	1	1	23	0	4
9	1	0	35	0	4

10 rows × 35 columns

3.3. Scaling

Standard Scaling

In [418…

```
std_scaler = StandardScaler()

bookings_data_standard = pd.DataFrame(std_scaler.fit_transform(bookings_d

print("Scaled Dataset Using Standard Scaler")

bookings_data_standard
```

Scaled Dataset Using Standard Scaler

Out [418…

	hotel	is_canceled	lead_time	stays_in_weekend_nights	stays_in_we
0	1.480287	-0.758663	-0.954599	-0.985063	
1	1.480287	-0.758663	-0.888162	-0.985063	
2	1.480287	-0.758663	-0.877090	-0.985063	
3	1.480287	-0.758663	-0.877090	-0.985063	
4	1.480287	-0.758663	-1.032108	-0.985063	
...	...	...	...	...	
103417	-0.675545	-0.758663	1.049564	1.319226	
103418	-0.675545	-0.758663	0.783819	1.319226	
103419	-0.675545	-0.758663	-0.799581	1.319226	
103420	-0.675545	-0.758663	-0.777435	1.319226	
103421	-0.675545	-0.758663	0.174819	1.319226	

103422 rows × 35 columns

3.4. Feature selection

In [419…

```
# The the variable to predict is 'is_canceled'

# Scaled Data: X

X = bookings_data_standard.drop(columns=['is_canceled'])

# Prediction Feature: is_canceled
y = bookings_data.is_canceled
```

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 [420…
```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
stratify=y, random_state=1)
```

Parameter explaination:

- test_size: This parameter defines the proportion of the dataset that will be allocated to the test set. 0.3 means 30% of dataset is for testing
- stratify: parameter ensures that the proportion of classes in the target variable y is preserved between the training and testing sets.
- random_state: This parameter sets the seed for the random number generator.

4.2. Model Training (10%)

In [421…
```
dt = DecisionTreeClassifier(criterion = 'entropy',random_state=1)
dt = dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
```

Parameter explaination:

- Criterion: The criterion parameter determines the function used to measure the quality of the split at each node.

Entropy: This measures the level of disorder or impurity in a dataset. The goal of a decision tree is to reduce entropy after each split.

- Random_state: parameter controls the randomness involved in the tree's construction, such as the random selection of features for splitting in some versions of decision trees.

Summary of Key Concepts

- Entropy: Measures disorder in the dataset. The algorithm aims to split the data to reduce entropy.
- Information Gain: The reduction in entropy from splitting a dataset based on a feature. The feature with the highest information gain is chosen for splitting.
- Random State: Controls the randomness in the tree-building process. By fixing it, you ensure reproducibility of results.

4.3. Model Evaluation (5%)

In [422…

```
data_accuracy = metrics.accuracy_score(y_test, y_pred)
print(f"Standardised Accuracy of the model is: {data_accuracy * 100:.2f}%
```

```
Standardised Accuracy of the model is: 81.88%
```

In [423…

```
print(metrics.classification_report(y_test,y_pred,output_dict=False))
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86     19693
           1       0.75      0.75      0.75     11334

    accuracy                           0.82     31027
   macro avg       0.80      0.80      0.80     31027
weighted avg       0.82      0.82      0.82     31027
```

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 [424…

```
# Variable importance in classifier
print("Variable importance in the classifier.")
pd.concat((pd.DataFrame(bookings_data_standard.iloc[:, 1:].columns, colum
['variable']),
pd.DataFrame(dt.feature_importances_, columns =
['importance'])),
axis = 1).sort_values(by='importance', ascending =
False)[:20]
```

```
Variable importance in the classifier.
```

Out[424…

	variable	importance
1	lead_time	0.217492
12	deposit_type	0.198596
14	adr	0.186101
3	stays_in_week_nights	0.061432
16	total_of_special_requests	0.045831
2	stays_in_weekend_nights	0.035354
28	market_segment_Online TA	0.033139
8	previous_cancellations	0.029603
10	assigned_room_type	0.024176
15	required_car_parking_spaces	0.023880
4	adults	0.019868
11	booking_changes	0.019653
9	reserved_room_type	0.019116
33	customer_type_Transient-Party	0.012430
0	is_canceled	0.011674
32	customer_type_Transient	0.009618
17	meal_BB	0.007682
24	market_segment_Corporate	0.005719
5	children	0.005684
19	meal_HB	0.005044

Commentary on the Reliability of the Model's Results Based on Feature Importance Scores

1. **Key Features with High Importance**

- **Lead Time (0.217)**: This feature has the highest importance, suggesting that the amount of time between the booking date and the arrival date plays a critical role in determining the model's output. This is plausible, especially in hotel bookings, as longer lead times may be associated with cancellations or different customer behaviors.
- **Deposit Type (0.198)**: The second most important feature is the type of deposit required at the time of booking. This makes sense because customers who pay a deposit upfront may be less likely to cancel, which could influence key outcomes

like booking cancellations.

- **Average Daily Rate (ADR) (0.186)**: This financial metric measures the revenue per day for occupied rooms and is another key factor. Higher ADR might indicate higher stakes for the booking, which could impact customer behavior, cancellation likelihood, or booking patterns.

The high importance of these three features reflects strong predictive value in the model. **Lead time, deposit type, and ADR** are all intuitive indicators of booking outcomes, suggesting that the model is correctly identifying the factors that are most relevant.

2. **Mid-Level Features with Moderate Importance**

- **Stays in Week Nights (0.061)** and **Stays in Weekend Nights (0.035)**: These features moderately influence the model, as the duration and timing of the stay can affect the likelihood of cancellations or other outcomes. This is reasonable, as weekday stays and weekend stays could have different cancellation rates or customer segments associated with them.
- **Special Requests (0.045)**: Customers making special requests might have different behaviors or expectations from their stay, so the moderate importance here seems reasonable.
- **Previous Cancellations (0.029)**: The model acknowledges that prior cancellation history has a small but notable impact on current booking behavior, which is reasonable. A history of cancellations may suggest higher cancellation risks.

3. **Low Importance Features**

- Features like **Children (0.0056)**, **Customer Type Transient (0.0096)**, and **Meal Plans** have relatively low importance scores. This might indicate that these features don't significantly impact the predictions made by the model, or their relationship with the target variable (e.g., booking outcomes) is weaker in the dataset. This could be because they do not vary much or because they are not strongly correlated with key booking behaviors.
- **Is Canceled (0.011)**: The fact that this feature (which might be a target variable in many hotel booking models) has such low importance suggests that the model may already be using lead indicators like lead time, ADR, and deposit type to predict the likelihood of cancellation. Thus, directly including "is_canceled" as a feature adds little additional value.

4. **Unimportant or Potentially Redundant Features**

- Features like **Market Segment (0.033)** and **Reserved Room Type (0.019)** have relatively low importance scores, suggesting they do not contribute much to the

model’s predictive power. This could indicate that other more important features are capturing similar information, making these features somewhat redundant.
- Some meal plan categories, such as **Meal HB (0.005)**, have very low importance, which is not surprising since meal plans might not significantly affect whether a booking is canceled or not.

5. **Overall Model Reliability**

- The feature importance distribution suggests that the model is **strongly relying on the most predictive features**, such as **lead time, deposit type, and ADR**, which intuitively makes sense for predicting hotel booking outcomes.
- However, some features with **very low importance** could be considered for removal in future iterations of model training to simplify the model and potentially improve generalization.
- The fact that features like **lead time** and **deposit type** align with common knowledge in the domain suggests that the model is identifying meaningful patterns, which supports the **reliability of the model**.

Improvements and Considerations

- **Feature Engineering**: You may explore interactions between key features (e.g., interaction between lead time and deposit type) or add new features if the model’s performance is unsatisfactory.
- **Feature Removal**: Low-importance features (e.g., meal plans, children) could be dropped to simplify the model without significantly affecting its predictive power.
- **Domain Expertise Alignment**: The high importance of financially driven features (deposit type, ADR) aligns with domain knowledge, suggesting the model's results are consistent with real-world expectations, enhancing confidence in the model’s reliability.

Conclusion

The model's results seem reliable based on the feature importance distribution, with **lead time, deposit type, and ADR** being the most critical features. These features are intuitive and relevant to hotel bookings, providing strong evidence that the model is capturing the main drivers of the outcome. However, the lower-importance features should be reviewed for potential removal to avoid overfitting and simplify the model further.