Analysis Data Hotel Bookings

Project by - Prabhat

This data is published on. https://www.sciencedirect.com/science/article/pii/S2352340918315191.

Data Preprocessing

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from google.colab import files
import io
%matplotlib inline
sns.set_style('whitegrid')
```

```
data = pd.read_csv('https://raw.githubusercontent.com/kevinasyraf/find-it-2020-dac/master/hotel_bookings.csv')
data.head()
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_w
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 rows × 32 columns

```
data.shape
```

```
(119390, 32)
```

```
# cek tipe data masing-masing atribut
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
```

```
# cek data yang hilang
data.isnull().sum()
```

```
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                             4
babies                               0
meal                                 0
country                            488
market_segment                       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
reservation_status                   0
reservation_status_date              0
dtype: int64
```

"We need to observe and fill in the missing data.

For the 'agent' attribute, it can be filled with 0.

In the 'company' attribute, there is a lot of missing data, so it's okay to delete this column.

For the 'country' attribute, it can be filled with 'Unknown' because the country of origin is unknown.

For the 'children' attribute, we can fill null data with the number 0."

```
data = data.drop('company', axis = 1)
```

```
data = data.fillna({
    'children' : 0,
    'agent' : 0,
    'country': 'Unknown',
})
```

```
"Ensure that no data is missing."
any(data.isna().sum())
```

```
False
```

```
#During the data observation, we found several rows where
#adult = 0, children = 0, and babies = 0. Therefore, rows containing data
#like this should be dropped because it is not possible to book a hotel with zero guests.

zero_guests = list(data.loc[data["adults"]
                  + data["children"]
                  + data["babies"]==0].index)
data.drop(data.index[zero_guests], inplace=True)
```

```
data.shape
```

```
(119210, 31)
```

EDA

There will be several questions to be answered in this Exploratory Data Analysis (EDA).

1. Detecting outliers

```
sns.boxplot(data=data, x = 'lead_time')
plt.show()

sns.boxplot(data=data, x = 'adr')
plt.show()
```

```
IQR_lt = data['lead_time'].quantile(0.75) -  data['lead_time'].quantile(0.25)
RUB = data['lead_time'].quantile(0.75) + 1.5*IQR_lt

data_no_outlier = data[data['lead_time'] <= RUB]
```

```
IQR_adr = data['adr'].quantile(0.75) -  data['adr'].quantile(0.25)
RUB = data['adr'].quantile(0.75) + 1.5*IQR_adr

data_no_outlier = data_no_outlier[data_no_outlier['adr'] <= RUB]
```

2.From which countries do the guests originate?

```
data_country = pd.DataFrame(data.loc[data['is_canceled'] != 1]['country'].value_counts())
data_country.index.name = 'country'
data_country.rename(columns={"country": "Number of Guests"}, inplace=True)
total_guests = data_country["Number of Guests"].sum()
data_country["Guests in %"] = round(data_country["Number of Guests"] / total_guests * 100, 2)
data_country.head(10) # The top 10 countries with the highest number of guests
```

country	Number of Guests	Guests in %
PRT	20977	27.97
GBR	9668	12.89
FRA	8468	11.29
ESP	6383	8.51
DEU	6067	8.09
IRL	2542	3.39
ITA	2428	3.24
BEL	1868	2.49
NLD	1716	2.29
USA	1592	2.12

From the data above, Portugal dominates as the home country of guests in the hotel.

```
import plotly.express as px
guest_map = px.choropleth(data_country,
                    locations=data_country.index,
                    color=data_country["Guests in %"],
                    hover_name=data_country.index,
                    color_continuous_scale=px.colors.sequential.Viridis,
                    title="The home countries of the hotel guests.")
guest_map.show()
```

The home countries of the hotel guests.

Based on the data visualization above, the hotel has been visited by tourists from almost every country. Countries shaded in white indicate that there have been no tourists from those countries visiting the hotel.

The European countries that frequently visit this hotel

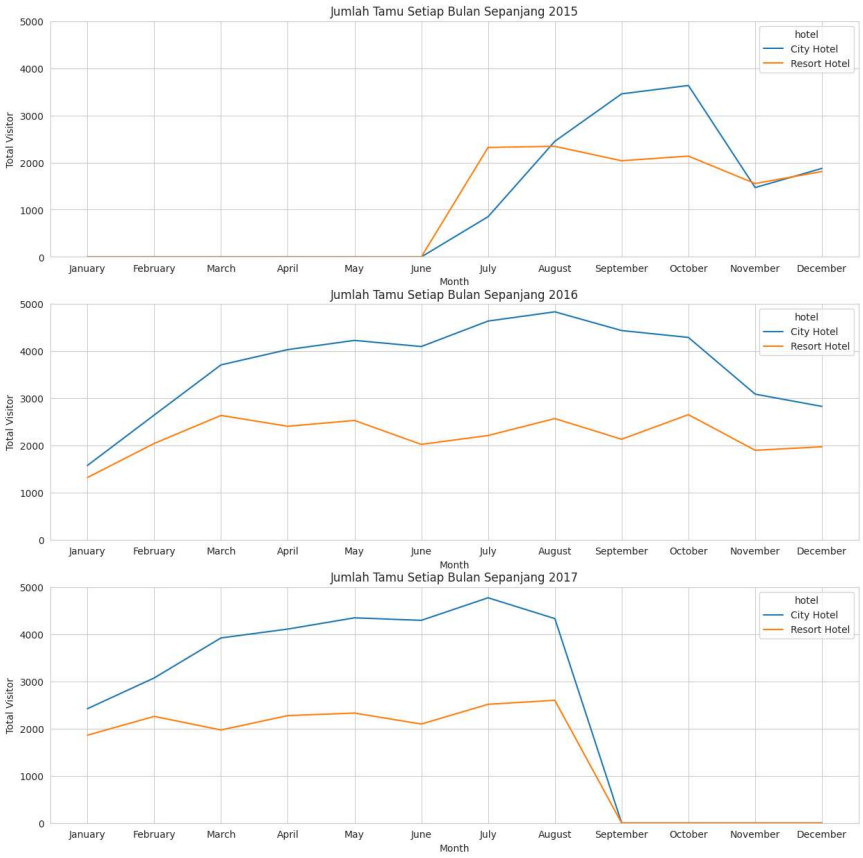
3. The number of guests per month based on the hotel for each year.

```
months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]
guest_data = data[data['is_canceled'] == 0].copy()
guests_monthly = guest_data[['hotel', 'arrival_date_year', 'arrival_date_month', 'adults', 'children', 'babies']].sort_values('arrival_date_
guests_monthly['total visitors'] = guests_monthly['adults'] + guests_monthly['children'] + guests_monthly['babies']
guests_monthly = guests_monthly.astype({'total visitors' : int})
guests_monthly = guests_monthly.drop(['adults', 'children', 'babies'], axis=1)
guests_monthly.head()
```

	hotel	arrival_date_year	arrival_date_month	total visitors
0	Resort Hotel	2015	July	2
43256	City Hotel	2015	September	2
43257	City Hotel	2015	September	2
43258	City Hotel	2015	September	2
43259	City Hotel	2015	September	2

```
guests_monthly['arrival_date_month'] = pd.Categorical(guests_monthly['arrival_date_month'], categories=months, ordered=True)
guests_monthly = guests_monthly.groupby(['hotel', 'arrival_date_year', 'arrival_date_month'], as_index = False).sum()

f, ax = plt.subplots(3,1,figsize=(15,15))
sns.lineplot(x = 'arrival_date_month', y="total visitors", hue="hotel", data=guests_monthly[guests_monthly['arrival_date_year'] == 2015],  c
sns.lineplot(x = 'arrival_date_month', y="total visitors", hue="hotel", data=guests_monthly[guests_monthly['arrival_date_year'] == 2016],  c
sns.lineplot(x = 'arrival_date_month', y="total visitors", hue="hotel", data=guests_monthly[guests_monthly['arrival_date_year'] == 2017],  c

ax[0].set(title="Jumlah Tamu Setiap Bulan Sepanjang 2015")
ax[0].set(xlabel="Month", ylabel="Total Visitor")
ax[0].set(ylim = (0,5000))

ax[1].set(title="Jumlah Tamu Setiap Bulan Sepanjang 2016")
ax[1].set(xlabel="Month", ylabel="Total Visitor")
ax[1].set(ylim = (0,5000))

ax[2].set(title="Jumlah Tamu Setiap Bulan Sepanjang 2017")
ax[2].set(xlabel="Month", ylabel="Total Visitor")
ax[2].set(ylim = (0,5000))

plt.show()
```

```
The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

<ipython-input-17-5128cee41ad3>:6: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

<ipython-input-17-5128cee41ad3>:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.
```



We can see from the attached graph that.

4. The total amount paid based on room type per night.

```
#Dividing the data based on hotels (Resort Hotel and City Hotel) that were not canceled.
rh = data_no_outlier[(data_no_outlier['hotel'] == 'Resort Hotel') & (data_no_outlier['is_canceled'] == 0)]
ch = data_no_outlier[(data_no_outlier['hotel'] != 'Resort Hotel') & (data_no_outlier['is_canceled'] == 0)]


# Calculating the Average Daily Rate (ADR) per individual (excluding infants).
rh['adr_pp'] = rh['adr'] / (rh['adults'] + rh['children'])
ch['adr_pp'] = ch['adr'] / (ch['adults'] + ch['children'])
```

```
<ipython-input-20-1074c44c0b0b>:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc

<ipython-input-20-1074c44c0b0b>:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
```

Due to the high number of visitors from Portugal, it is highly likely that the currency used is Euro (€).

```
print(f"""
    The average price paid per person per night is:
    Resort Hotel: {rh['adr_pp'].mean():.2f} €
    City Hotel: {ch['adr_pp'].mean():.2f} €""")
    )
```

```
    The average price paid per person per night is:
    Resort Hotel: 44.50 €
    City Hotel: 58.82 €
```

```
full_data_guests = data.copy()
full_data_guests = full_data_guests.loc[full_data_guests['is_canceled'] == 0]
full_data_guests['adr_pp'] = full_data_guests['adr'] / (full_data_guests['adults'] + full_data_guests['children'])
room_prices = full_data_guests[['hotel', 'reserved_room_type', 'adr_pp']].sort_values("reserved_room_type")
```

```
<ipython-input-22-515fb0a70c7f>:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
```

```
plt.figure(figsize=(10,5))
sns.barplot(x='reserved_room_type', y='adr_pp', hue='hotel', data=room_prices, hue_order=['City Hotel', 'Resort Hotel'], palette='pastel')
plt.title('The room type prices per night per person based on the hotel.', fontsize=16)
plt.xlabel('Room Types.', fontsize = 16)
plt.ylabel('Prices.(€)', fontsize = 16)
plt.show()
```

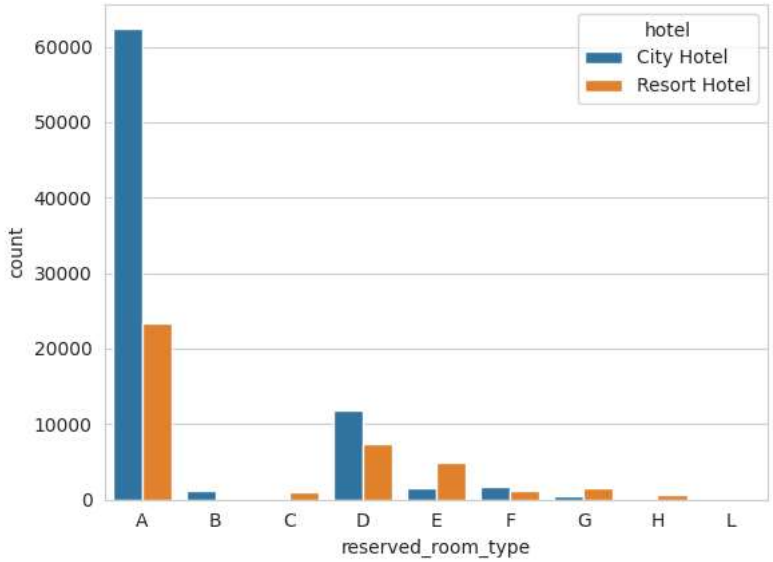
5. The most frequently booked room type.

```
print('Frekuensi pemesanan tiap-tiap tipe kamar pada CITY HOTEL')
print(data[(data['hotel'] == 'City Hotel')]['reserved_room_type'].value_counts())
print()
print('Frekuensi pemesanan tiap-tiap tipe kamar pada RESORT HOTEL')
print(data[data['hotel'] != 'City Hotel']['reserved_room_type'].value_counts())
```

```
Frekuensi pemesanan tiap-tiap tipe kamar pada CITY HOTEL
A    62484
D    11747
F     1788
E     1537
B     1112
G      482
C       13
Name: reserved_room_type, dtype: int64

Frekuensi pemesanan tiap-tiap tipe kamar pada RESORT HOTEL
A    23389
D     7432
E     4982
G     1610
F     1106
C      918
H      601
L        6
B        3
Name: reserved_room_type, dtype: int64
```

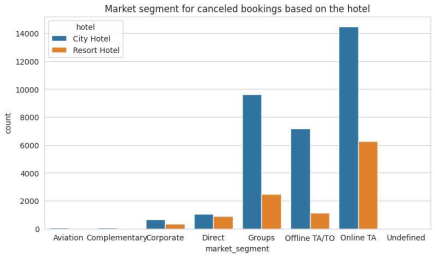
```
sns.countplot(x = 'reserved_room_type', data = data.sort_values('reserved_room_type'), hue='hotel')
```

```
<Axes: xlabel='reserved_room_type', ylabel='count'>
```



6.Comparison of "market segment" based on the hotel.

```
data_canceled = data[data['is_canceled'] == 1].sort_values('market_segment')
data_not_canceled = data[data['is_canceled'] == 0].sort_values('market_segment')
f, ax = plt.subplots(1,2,figsize=(20,5))
sns.countplot(data=data_canceled, x= 'market_segment', hue='hotel', ax =ax[0])
sns.countplot(data=data_not_canceled, x= 'market_segment', hue='hotel', ax =ax[1])
ax[0].set(title='Market segment for canceled bookings based on the hotel')
ax[1].set(title='Market segment for canceled bookings based on the hotel')
plt.show()
```



7. Correlation of each feature available.

```
plt.figure(figsize=(20, 20))
sns.heatmap(data.corr(), annot=True)
```

```
<ipython-input-28-4b0845da6f75>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version,

<Axes: >
```

"Here, we can observe a strong correlation between certain factors:

1] lead_time with is_cancelled The likelihood of booking cancellations is significantly higher for customers who make reservations well in advance. Plans made far in advance are more likely to change due to unforeseen events. Another reason customers might cancel bookings made far in advance (often facilitated by free cancellation options) could be due to changes in plans, natural disasters, or sudden illness.

2] previous_bookings_not_cancelled with is_repeated From the data, we can conclude that a significant number of customers who place repeat orders do not cancel their previous bookings. This suggests that these customers are satisfied with the hotel's services. Consequently, reducing the cancellation rate could potentially lead to an increased number of repeat bookings.

Understanding these correlations can provide insights into customer behavior and help in developing strategies to minimize cancellations and encourage repeat bookings."

8. The percentage of adults who bring children versus those who do not.

```
adult_only = data[(data['adults'] != 0) & (data['children'] == 0) & (data['babies'] == 0)].sort_values('reserved_room_type')
adult_child = data[(data['adults'] != 0) & (data['children'] != 0) | (data['babies'] != 0)].sort_values('reserved_room_type')
```



```
percentage = [(len(adult_only)/(len(adult_only) + len(adult_child)))*100, (len(adult_child)/(len(adult_only) + len(adult_child)))*100]
labels = 'Not Bringing Children', 'Bringing Children'

f, ax = plt.subplots(figsize=(7,7))
ax.pie(percentage, labels = labels, autopct='%1.1f%%' , startangle = 180)
ax.axis('equal')

ax.set_title('The percentage of adults who bring children and those who do not.', fontsize=16)
plt.show()
```

The percentage of adults who bring children and those who do not.

9. How many bookings were canceled?

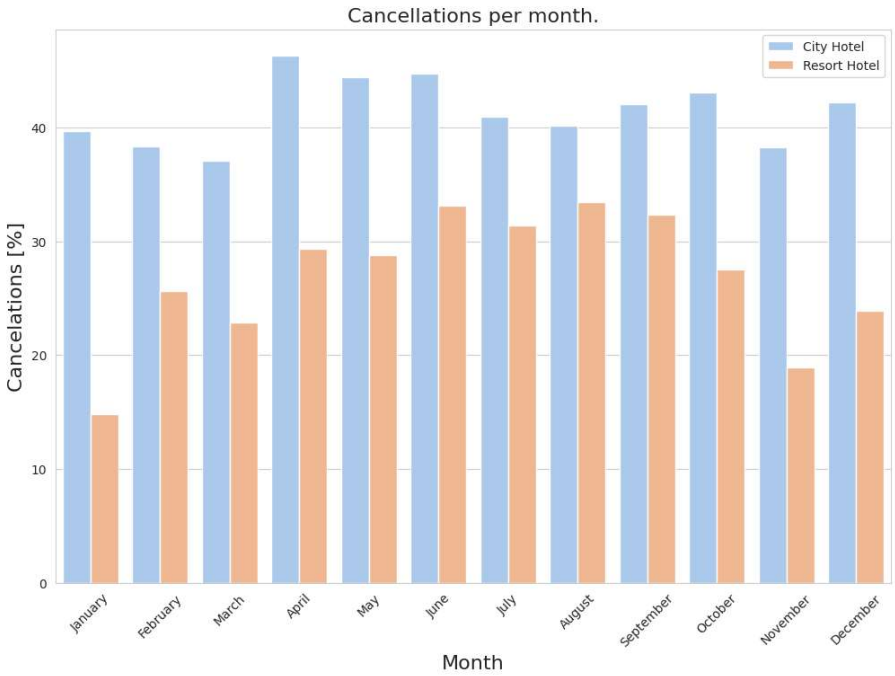
From this, we can see that there are more cancellations during the booking process in the city hotel compared to the resort hotel. This may occur because people booking a resort hotel, located in a quieter and more remote area, likely have clear intentions for a vacation. This contrasts with the booking process in city hotels.


```
total_cancelations = data['is_canceled'].sum()
rh_cancelations = data.loc[data["hotel"] == "Resort Hotel"]["is_canceled"].sum()
ch_cancelations = data.loc[data["hotel"] == "City Hotel"]["is_canceled"].sum()

# mencari persentase
rel_cancel = (total_cancelations / data.shape[0]) * 100
rh_rel_cancel = (rh_cancelations / data.loc[data["hotel"] == "Resort Hotel"].shape[0]) * 100
ch_rel_cancel = (ch_cancelations / data.loc[data["hotel"] == "City Hotel"].shape[0]) * 100

print(f"The number of bookings that were canceled.: {total_cancelations:} ({rel_cancel:.0f} %)")
print(f"The number of bookings canceled for the Resort hotel.: {rh_cancelations:} ({rh_rel_cancel:.0f} %)")
print(f"The number of bookings canceled for the City hotel.: {ch_cancelations:} ({ch_rel_cancel:.0f} %)")
```

```
The number of bookings that were canceled.: 44199 (37 %)
The number of bookings canceled for the Resort hotel.: 11120 (28 %)
The number of bookings canceled for the City hotel.: 33079 (42 %)
```

10. How many total cancellations are there each month?

```
res_book_per_month = data.loc[(data["hotel"] == "Resort Hotel")].groupby("arrival_date_month")["hotel"].count()
res_cancel_per_month = data.loc[(data["hotel"] == "Resort Hotel")].groupby("arrival_date_month")["is_canceled"].sum()

cty_book_per_month = data.loc[(data["hotel"] == "City Hotel")].groupby("arrival_date_month")["hotel"].count()
cty_cancel_per_month = data.loc[(data["hotel"] == "City Hotel")].groupby("arrival_date_month")["is_canceled"].sum()

res_cancel_data = pd.DataFrame({"Hotel": "Resort Hotel",
                                "Month": list(res_book_per_month.index),
                                "Bookings": list(res_book_per_month.values),
                                "Cancelations": list(res_cancel_per_month.values)})

cty_cancel_data = pd.DataFrame({"Hotel": "City Hotel",
                                "Month": list(cty_book_per_month.index),
                                "Bookings": list(cty_book_per_month.values),
                                "Cancelations": list(cty_cancel_per_month.values)})

full_cancel_data = pd.concat([res_cancel_data, cty_cancel_data], ignore_index=True)
full_cancel_data["cancel_percent"] = full_cancel_data["Cancelations"] / full_cancel_data["Bookings"] * 100

ordered_months = ["January", "February", "March", "April", "May", "June",
          "July", "August", "September", "October", "November", "December"]
full_cancel_data["Month"] = pd.Categorical(full_cancel_data["Month"], categories=ordered_months, ordered=True)

plt.figure(figsize=(12, 8))
sns.barplot(x = "Month", y = "cancel_percent" , hue="Hotel",
            hue_order = ["City Hotel", "Resort Hotel"], data=full_cancel_data, palette = 'pastel')
plt.title("Cancellations per month.", fontsize=16)
plt.xlabel("Month", fontsize=16)
plt.xticks(rotation=45)
plt.ylabel("Cancelations [%]", fontsize=16)
plt.legend(loc="upper right")
plt.show()
```

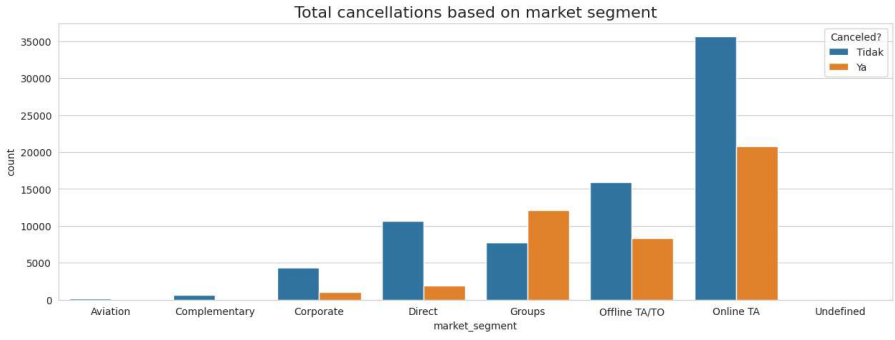


BACOT DI SINI

11. Number of Cancellations based on market_segment

It can be observed from the graph that the highest number of cancellations occurs in the online travel agent segment, likely due to the convenience offered by online travel agents in both booking and canceling reservations. This makes potential customers more inclined to book hotels without second thoughts, given the ease and often free cancellation options. The combination of these factors contributes to the high cancellation rate in reservations made through online travel agents.

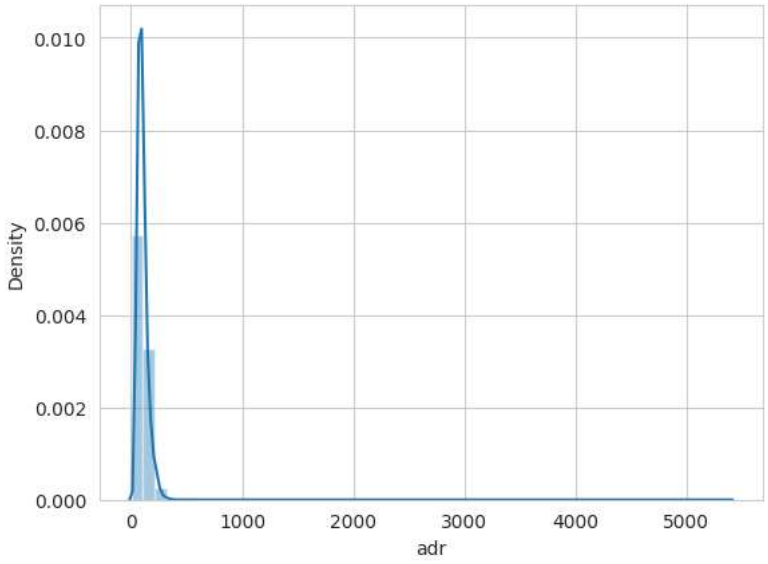
```
plt.figure(figsize=(15,5))
sns.countplot(x='market_segment', data=data.sort_values('market_segment'), hue = 'is_canceled')

plt.legend(title='Canceled?', loc='best', labels=['Tidak', 'Ya'])
plt.title('Total cancellations based on market segment', size = 16)
plt.show()
```



12. Distribution of ADR (Average Daily Rate).

```
sns.distplot(data[data['adr'] > 0]['adr'])
```

```
<ipython-input-36-fd390d59bf8e>:1: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<Axes: xlabel='adr', ylabel='Density'>
```



Machine Learning Modelling

After observation, it was found that there are some features considered unnecessary. Therefore, the selection is done manually.

```
from sklearn.model_selection import train_test_split, KFold, cross_validate, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier
from xgboost import XGBClassifier, plot_importance, DMatrix, train
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
```

```
dtrain = pd.read_csv('https://raw.githubusercontent.com/kevinasyraf/find-it-2020-dac/master/hotel_bookings.csv')
```

```
nan_replacements = {"children:": 0.0,"country": "Unknown", "agent": 0, "company": 0}
dtrain = dtrain.fillna(nan_replacements)

# "meal" contains values "Undefined", which is equal to SC.
dtrain["meal"].replace("Undefined", "SC", inplace=True)
```

```
dtrain=dtrain.drop(['company'],axis=1)
dtrain=dtrain.dropna(axis=0)
```

```
dtrain.isna().sum()
```

```
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
babies                            0
meal                              0
country                           0
market_segment                    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                             0
days_in_waiting_list              0
customer_type                     0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
reservation_status                0
reservation_status_date           0
dtype: int64
```

Label Encoding

Each categorical data will be encoded into a numerical format. This is done to facilitate training in machine learning since machines are more adept at processing numerical data than strings.

```
# hotel
dtrain['hotel']=dtrain['hotel'].map({'Resort Hotel':0,'City Hotel':1})
dtrain['hotel'].unique()
```

```
array([0, 1])
```

```
# arrival_date_month
dtrain['arrival_date_month'] = dtrain['arrival_date_month'].map({'July':7,'August':8,'September':9,'October':10
                                                                 ,'November':11,'December':12,'January':1,'February':2,'March':3,
                                                                 'April':4,'May':5,'June':6})
dtrain['arrival_date_month'].unique()
```

```
array([ 7,  8,  9, 10, 11, 12,  1,  2,  3,  4,  5,  6])
```

Now we will leverage the LabelEncoder library, which is much faster and more efficient than defining the values of categorical data one by one.

```
label_encoder = LabelEncoder()
```

```
dtrain['meal']=label_encoder.fit_transform(dtrain['meal'])
dtrain['meal'].unique()
```

```
array([0, 1, 2, 3])
```

```
dtrain['country']=label_encoder.fit_transform(dtrain['country'])
dtrain['market_segment']=label_encoder.fit_transform(dtrain['market_segment'])
dtrain['distribution_channel']=label_encoder.fit_transform(dtrain['distribution_channel'])
dtrain['reserved_room_type']=label_encoder.fit_transform(dtrain['reserved_room_type'])
dtrain['assigned_room_type']=label_encoder.fit_transform(dtrain['assigned_room_type'])
dtrain['deposit_type']=label_encoder.fit_transform(dtrain['deposit_type'])
dtrain['customer_type']=label_encoder.fit_transform(dtrain['customer_type'])
dtrain['reservation_status']=label_encoder.fit_transform(dtrain['reservation_status'])
dtrain['reservation_status_date']=label_encoder.fit_transform(dtrain['reservation_status_date'])
```

```
dtrain.head()
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_we
0	0	0	342	2015	7	
1	0	0	737	2015	7	
2	0	0	7	2015	7	
3	0	0	13	2015	7	

Feature Extraction

```
# Gathering which feature is more important.....using corr() function
correlation=dtrain.corr()['is_canceled']
correlation.abs().sort_values(ascending=False)
```

```
is_canceled                       1.000000
reservation_status                0.917191
deposit_type                      0.468665
lead_time                         0.293177
country                           0.264709
total_of_special_requests         0.234706
required_car_parking_spaces       0.195492
assigned_room_type                0.176025
distribution_channel              0.167544
booking_changes                   0.144371
reservation_status_date           0.143258
hotel                             0.136505
previous_cancellations            0.110140
is_repeated_guest                 0.084788
customer_type                     0.068210
reserved_room_type                0.061284
adults                            0.059990
market_segment                    0.059314
previous_bookings_not_canceled    0.057355
days_in_waiting_list              0.054193
adr                               0.047622
agent                             0.046503
babies                            0.032488
stays_in_week_nights              0.024771
arrival_date_year                 0.016732
meal                              0.015671
arrival_date_month                0.011002
arrival_date_week_number          0.008132
arrival_date_day_of_month         0.006084
children                          0.005048
stays_in_weekend_nights           0.001783
Name: is_canceled, dtype: float64
```

```
cols=['arrival_date_day_of_month','children',
     'arrival_date_week_number','stays_in_week_nights','reservation_status']
dtrain=dtrain.drop(cols,axis=1)
dtrain.head(5)
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	stays_in_weeken
0	0	0	342	2015	7	
1	0	0	737	2015	7	
2	0	0	7	2015	7	
3	0	0	13	2015	7	
4	0	0	14	2015	7	

5 rows × 26 columns

```
dtrain.shape
```

```
(119386, 26)
```

Building the model.

```
y=dtrain['is_canceled'].values
x=dtrain.drop(['is_canceled'],axis=1).values


# dataset split.
train_size=0.80
test_size=0.20
seed=5

x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=train_size,test_size=test_size,random_state=seed)


ensembles=[]
ensembles.append(('scaledRFC',Pipeline([('scale',StandardScaler()),('rf',RandomForestClassifier(n_estimators=10))])))

results=[]
names=[]
for name,model in ensembles:
    fold = KFold(n_splits=10,random_state=None)
    result = cross_val_score(model,x_train,y_train,cv=fold,scoring='accuracy')
    results.append(result)
    names.append(name)
    msg="%s : %f (%f)"%(name,result.mean(),result.std())
    print(msg)
```

```
 scaledRFC : 0.930634 (0.002952)
```

Based on the training model above using RFC and StandardScaler, the accuracy result obtained for the Random Forest Classifier is **0.931576 (0.002612)**

```
# Random Forest Classifier Tuning
from sklearn.model_selection import GridSearchCV

scaler=StandardScaler().fit(x_train)
rescaledx=scaler.transform(x_train)

n_estimators=[10,20,30,40,50]

param_grid=dict(n_estimators=n_estimators)

model=RandomForestClassifier()

fold=KFold(n_splits=10,random_state=None)

grid=GridSearchCV(estimator=model,param_grid=param_grid,scoring='accuracy',cv=fold)
grid_result=grid.fit(rescaledx,y_train)

print("Best: %f using %s "%(grid_result.best_score_,grid_result.best_params_))
```

```
 Best: 0.940078 using {'n_estimators': 40}
```

Random Forest

Based on the results of the two tuning steps above, Random Forest will be used to make predictions from our data.

```
from sklearn.metrics import confusion_matrix

scaler=StandardScaler().fit(x_train)
scaler_x=scaler.transform(x_train)
model=RandomForestClassifier(n_estimators=40)
model.fit(scaler_x,y_train)

#Transform the validation test set data
scaledx_test=scaler.transform(x_test)
y_pred=model.predict(scaledx_test)

accuracy_mean=accuracy_score(y_test,y_pred)
accuracy_matric=confusion_matrix(y_test,y_pred)
print(accuracy_mean)
print(accuracy_matric)
```

```
 0.9412429851746378
 [[14662   340]
```

```
[ 1063  7813]]
```

```
# Predicting the entire dataset by first transforming the data.
y_pred = model.predict(scaler.transform(x))
print(accuracy_score(y, y_pred))
```

```
0.9879466604124437
```

And we successfully achieved an accuracy of 98.87% :D

```
predicted_data = pd.DataFrame({'is_canceled' : pd.Series(y_pred)})
predicted_data.head(10)
```

	is_canceled
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	1

```
# It's time to export the prediction results to an Excel file
predicted_data.to_excel('results.xlsx', index=False)
```

```
predicted_data.shape
```

```
(119386, 1)
```