databricks GAL-EDA & Modelling_Nov2023

(https://databricks.com)

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tabulate import tabulate
import calendar
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
```

Data Load

```
# Read the CSV file into a Pandas DataFrame
df = spark.read.format('csv').options(header='true').load('dbfs:/FileStore/df_2021_filter.csv').toPandas()

# Display the DataFrame
display(df)
```

Table

	car_park_prebook_description	carpark_type_name	booking_summary_source_booking_num	booking_date_date_value	booked_entry_date_date_value	booked_exit_date_date_value	actual_e
1	ST Long Stay	Prebook	DW00395745QAI	2021-10-27	2022-07-22	2022-08-03	2022-07
2	ST Long Stay	Prebook	DW01373481ASH	2022-05-17	2022-05-25	2022-05-29	2022-05
3	ST Long Stay	Prebook	DW01046827CRI	2022-04-04	2022-04-24	2022-05-01	2022-04
4	ST Long Stay	Prebook	DW00631505CHR	2022-01-24	2022-02-04	2022-02-06	2022-02
5	ST Long Stay	Prebook	DW01346001WAQ	2022-05-14	2022-05-22	2022-05-26	2022-05
6	ST Long Stay	Prebook	DW01506960PHI	2022-06-01	2022-06-05	2022-06-13	2022-06
7	ST Long Stay	Prebook	DW03496401LIS	2023-02-25	2023-07-31	2023-08-15	2023-07

7,198 rows | Truncated data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 467660 entries, 0 to 467659
Data columns (total 27 columns):
 #   Column                                 Non-Null Count   Dtype
---  ------                                 --------------   -----
 0   car_park_prebook_description           467660 non-null  object
 1   carpark_type_name                      467660 non-null  object
 2   booking_summary_source_booking_num     467660 non-null  object
 3   booking_date_date_value                467660 non-null  object
 4   booked_entry_date_date_value           467660 non-null  object
 5   booked_exit_date_date_value            467660 non-null  object
 6   actual_entry_date_date_value           467660 non-null  object
 7   actual_exit_date_date_value            467660 non-null  object
 8   booking_date_calendar_year_week        467660 non-null  object
 9   booked_entry_date_calendar_year_week   467660 non-null  object
 10  booked_exit_date_calendar_year_week    467660 non-null  object
 11  booking_date_calendar_by_day_month     467660 non-null  object
 12  booked_entry_date_calendar_by_day_month 467660 non-null  object
 13  booked_exit_date_calendar_by_day_month 467660 non-null  object
 14  booking_date_phase                     467660 non-null  object
 15  booked_entry_date_phase                467660 non-null  object
```

```
# Print the data types for all variables
df.dtypes
```

```
Out[4]: car_park_prebook_description              object
carpark_type_name                         object
booking_summary_source_booking_num        object
booking_date_date_value                   object
booked_entry_date_date_value              object
booked_exit_date_date_value               object
actual_entry_date_date_value              object
actual_exit_date_date_value               object
booking_date_calendar_year_week           object
booked_entry_date_calendar_year_week      object
booked_exit_date_calendar_year_week       object
booking_date_calendar_by_day_month        object
booked_entry_date_calendar_by_day_month   object
booked_exit_date_calendar_by_day_month    object
booking_date_phase                        object
booked_entry_date_phase                   object
booked_exit_date_phase                    object
booking_detail_staydays                   object
channel_long_description                  object
actual_exit_date_week_day_name            object
actual_entry_date_week_day_name           object
```

Data Transformation

```
# Convert date columns to datetime format
df['booking_date_date_value'] = pd.to_datetime(df['booking_date_date_value'])
# Convert date columns to datetime format
df['actual_entry_date_date_value'] = pd.to_datetime(df['actual_entry_date_date_value'])
# Convert date columns to datetime format
df['actual_exit_date_date_value'] = pd.to_datetime(df['actual_exit_date_date_value'])
# Convert Stay days to integer
df['booking_detail_staydays'] = df['booking_detail_staydays'].astype(int)
#Convert Gross Amount to Float
df['booking_detail_source_gross_amount'] = df['booking_detail_source_gross_amount'].astype(float)
# Convert booking_detail_lead_time to integer
df['booking_detail_lead_time'] = df['booking_detail_lead_time'].astype(int)
# Convert Capacity to integer
df['car_park_carpark_capacity'] = df['car_park_carpark_capacity'].astype(int)
```

Add Avg_GrossAmount_per_StayDay

```
# Calculate 'Avg_GrossAmount_per_StayDay' and add it as a new column
df['Avg_GrossAmount_per_StayDay'] = df['booking_detail_source_gross_amount'] / df['booking_detail_staydays']
```

```
# Print the data types for all variables
df.dtypes
```

```
Out[7]: car_park_prebook_description                    object
carpark_type_name                              object
booking_summary_source_booking_num             object
booking_date_date_value                datetime64[ns]
booked_entry_date_date_value                   object
booked_exit_date_date_value                    object
actual_entry_date_date_value           datetime64[ns]
actual_exit_date_date_value            datetime64[ns]
booking_date_calendar_year_week                object
booked_entry_date_calendar_year_week           object
booked_exit_date_calendar_year_week            object
booking_date_calendar_by_day_month             object
booked_entry_date_calendar_by_day_month        object
booked_exit_date_calendar_by_day_month         object
booking_date_phase                             object
booked_entry_date_phase                        object
booked_exit_date_phase                         object
booking_detail_staydays                         int64
channel_long_description                       object
actual_exit_date_week_day_name                 object
actual_entry_date_week_day_name                object
```

ST Long Stay Data from Jan 2023 >> Prebook with Confirmed Booking Status, Gross Amount >0 and Stay Days > 0

```
# Filter the df_2023 data further : "ST Long Stay" Data from Jan 2023 >> Prebook with Confirmed Booking Status, Gross Amount >0 and Stay Days > 0
df = df[df["booking_detail_staydays"] > 0]
```

Descriptive Statistics

With Descriptive Stats, identified Duplicate values in Booking No.

Upon manual investigation, its observed that that, this is due to multiple add-on purchases on the same booking-id.

```
df.describe(include="all")
```

	car_park_prebook_description	carpark_type_name	booking_summary_source_booking_num	booking_date_date_value	booked_entry_date_date_value	booked_exit_date_date_value	actual_entry_date_date_value	actual_exit_d
count	426366	426366	426366	426366	426366	426366	426366	
unique	1	2	422950	882	1146	1151	1146	
top	ST Long Stay	Prebook	DWC02697751JAM	2023-08-24 00:00:00	2023-09-01	2023-01-02	2022-10-22 00:00:00	202
freq	426366	422810	3	1806	1678	1568	1681	
first	NaN	NaN	NaN	2021-01-12 00:00:00	NaN	NaN	2021-06-11 00:00:00	202
last	NaN	NaN	NaN	2023-10-03 00:00:00	NaN	NaN	2024-09-21 00:00:00	202
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

Descriptive Statistics on Numeric Variables

```
df.describe()
```

	booking_detail_staydays	car_park_carpark_capacity	booking_detail_source_gross_amount	booking_detail_lead_time	Avg_GrossAmount_per_StayDay
count	426366.000000	426366.0	426366.000000	426366.000000	426366.000000
mean	7.183549	8316.0	86.563271	27.295077	13.139161
std	4.320988	0.0	37.647554	44.727845	3.544702
min	1.000000	8316.0	10.000000	0.000000	0.484419
25%	4.000000	8316.0	56.000000	3.000000	10.666667
50%	7.000000	8316.0	80.000000	11.000000	13.071429
75%	8.000000	8316.0	109.000000	31.000000	15.500000
max	353.000000	8316.0	655.000000	357.000000	42.000000

To circumvent duplicates, created New Table (group_df) which aggregates Booking No. whilst keeping relevant columns

-Total New Records : ~423K

```
# Group by 'booking_summary_source_booking_num' and aggregate the data
group_df = df.groupby('booking_summary_source_booking_num').agg({
    'carpark_type_name': 'first',  # Use 'first' to get the first value in the group
    'booking_date_date_value': 'first',
    'booked_entry_date_date_value': 'first',
    'booked_exit_date_date_value': 'first',
    'booking_date_phase' : 'first',
    'booked_entry_date_phase' : 'first',
    'booked_exit_date_phase' : 'first',
    'channel_long_description': 'first',
    'actual_entry_date_week_day_name': 'first',
    'actual_exit_date_week_day_name': 'first',
    'booking_date_calendar_year_week': 'first',
    'booking_date_calendar_by_day_month': 'first',
    'booked_entry_date_calendar_by_day_month': 'first',
    'booked_exit_date_calendar_by_day_month': 'first',
    'booking_summary_outbound_flight': 'first',
    'booking_detail_staydays': 'sum',
    'car_park_carpark_capacity': 'mean',
    'booking_detail_lead_time': 'mean',
    'booking_detail_source_gross_amount': 'sum',
    'Avg_GrossAmount_per_StayDay': 'sum',
}).reset_index()

# Rename the columns for clarity
group_df.rename(columns={
    'carpark_type_name': 'CarPark_Type',
    'booking_date_date_value': 'Booking_Date',
    'booked_entry_date_date_value': 'Entry_Date',
    'booked_exit_date_date_value': 'Exit_Date',
    'booking_date_phase': 'Booking_Season',
    'booked_entry_date_phase' : 'Entry_Season',
    'booked_exit_date_phase' : 'Exit_Season',
    'channel_long_description': 'Channel',
    'actual_entry_date_week_day_name': 'Actual_Entry_Day',
    'actual_exit_date_week_day_name': 'Actual_Exit_Day',
    'booking_date_calendar_year_week': 'Booking_Week',
    'booking_date_calendar_by_day_month': 'Booking_Month_No',
    'booked_entry_date_calendar_by_day_month': 'Booking_Entry_Month_No',
    'booked_exit_date_calendar_by_day_month': 'Booking_Exit_Month_No',
    'booking_summary_outbound_flight': 'Outbound_Flight',
    'booking_detail_staydays': 'Total_Staydays',
    'car_park_carpark_capacity': 'Carpark_Capacity',
    'booking_detail_lead_time': 'Booking_Lead_Time',
    'booking_detail_source_gross_amount': 'Total_Gross_Amount',
    'Avg_GrossAmount_per_StayDay': 'Avg_GrossAmount_per_StayDay',
}, inplace=True)
```

```
group_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 422950 entries, 0 to 422949
Data columns (total 21 columns):
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   booking_summary_source_booking_num  422950 non-null  object
 1   CarPark_Type                        422950 non-null  object
 2   Booking_Date                        422950 non-null  datetime64[ns]
 3   Entry_Date                          422950 non-null  object
 4   Exit_Date                           422950 non-null  object
 5   Booking_Season                      422950 non-null  object
 6   Entry_Season                        422950 non-null  object
 7   Exit_Season                         422950 non-null  object
 8   Channel                             422950 non-null  object
 9   Actual_Entry_Day                    422950 non-null  object
 10  Actual_Exit_Day                     422950 non-null  object
 11  Booking_Week                        422950 non-null  object
 12  Booking_Month_No                    422950 non-null  object
 13  Booking_Entry_Month_No              422950 non-null  object
 14  Booking_Exit_Month_No               422950 non-null  object
 15  Outbound_Flight                     39091 non-null   object
```

```
group_df.describe(include="all")
```

	booking_summary_source_booking_num	CarPark_Type	Booking_Date	Entry_Date	Exit_Date	Booking_Season	Entry_Season	Exit_Season	Channel	Actual_Entry_Day	Actual_Exit_Day	Booking_Week	Booking_Month_No
count	422950	422950	422950	422950	422950	422950	422950	422950	422950	422950	422950	422950	422950
unique	422950	2	882	1146	1151	3	3	3	6	7	7	137	12
top	DW00064596JOH	Prebook	2023-08-24 00:00:00	2023-09-01	2023-01-02	Summer Peak	Summer Peak	Summer Peak	Direct Web	Friday	Sunday	202331	202308
freq	1	420866	1774	1664	1557	173162	190901	195907	408068	78693	81889	7799	5342
first	NaN	NaN	2021-01-12 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
last	NaN	NaN	2023-10-03 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
group_df.describe()
```

	Total_Staydays	Carpark_Capacity	Booking_Lead_Time	Total_Gross_Amount	Avg_GrossAmount_per_StayDay
count	422950.000000	422950.0	422950.000000	422950.000000	422950.000000
mean	7.241568	8316.0	27.431289	87.262408	13.245281
std	4.318966	0.0	44.774042	37.340297	3.633934
min	1.000000	8316.0	0.000000	10.000000	0.484419
25%	4.000000	8316.0	3.000000	58.250000	10.800000
50%	7.000000	8316.0	11.000000	81.000000	13.111111
75%	8.000000	8316.0	31.000000	111.000000	15.637500
max	353.000000	8316.0	357.000000	655.000000	44.200000

Data Output from New table (group_df) is stored in DBFS-Filestore location

Total Variables: 14

Total Records: 422950

```
#outname = 'Data_Processed_ST_LongStay.csv'
#outdir = '/dbfs/FileStore/'
#group_df.to_csv(outdir+outname, index=False, encoding="utf-8")
```

Add new features

Add New Features : Month_Name, Year

```
# Add week number, month number, and month name columns
group_df['Month_Name'] = group_df['Booking_Date'].dt.strftime('%b')
group_df['Year'] = group_df['Booking_Date'].dt.year

# Display the DataFrame with the added columns
print(group_df[['Month_Name', 'Year']])
```

```
     Month_Name  Year
0           Jan  2021
1           Jan  2021
2           Feb  2021
3           Feb  2021
4           Feb  2021
```

```
...          ...   ...
422945       Oct  2023
422946       Oct  2023
422947       Oct  2023
422948       Oct  2023
422949       Mar  2023

[422950 rows x 2 columns]
```

```
  # Remove values from "T" onwards in the Booking_Date column
  group_df['Booking_Date'] = group_df['Booking_Date'].dt.strftime('%Y-%m-%d')


  # Display the DataFrame with the modified 'Booking_Date' column
  print(group_df[['Booking_Date', 'Month_Name', 'Year']])
```

```
      Booking_Date Month_Name  Year
0       2021-01-22        Jan  2021
1       2021-01-23        Jan  2021
2       2021-02-23        Feb  2021
3       2021-02-23        Feb  2021
4       2021-02-25        Feb  2021
...            ...        ...   ...
422945  2023-10-03        Oct  2023
422946  2023-10-03        Oct  2023
422947  2023-10-03        Oct  2023
422948  2023-10-03        Oct  2023
422949  2023-03-22        Mar  2023

[422950 rows x 3 columns]
```

```
  display(group_df)
```

Table

	booking_summary_source_booking_num	CarPark_Type	Booking_Date	Entry_Date	Exit_Date	Booking_Season	Entry_Season	Exit_Season	Channel	Actual_Entry_Day
1	DW00064596JOH	Prebook	2021-01-22	2021-08-24	2021-08-31	Winter Shoulder	Summer Peak	Summer Peak	Direct Web	Tuesday
2	DW00064666DAV	Prebook	2021-01-23	2021-08-13	2021-08-27	Winter Shoulder	Summer Peak	Summer Peak	Direct Web	Friday
3	DW00069398ALA	Prebook	2021-02-23	2021-06-22	2021-07-06	Winter Shoulder	Summer Shoulder	Summer Peak	Direct Web	Tuesday
4	DW00069400ALA	Prebook	2021-02-23	2021-06-22	2021-07-06	Winter Shoulder	Summer Shoulder	Summer Peak	Direct Web	Tuesday
5	DW00070093LUK	Prebook	2021-02-25	2021-06-14	2021-06-30	Winter Shoulder	Summer Shoulder	Summer Shoulder	Direct Web	Monday
6	DW00070739LAW	Prebook	2021-02-26	2021-10-04	2021-10-19	Winter Shoulder	Summer Peak	Summer Peak	Direct Web	Monday
7	DW00070742CYN	Prebook	2021-02-26	2021-06-11	2021-06-14	Winter Shoulder	Summer Shoulder	Summer Shoulder	Direct Web	Friday

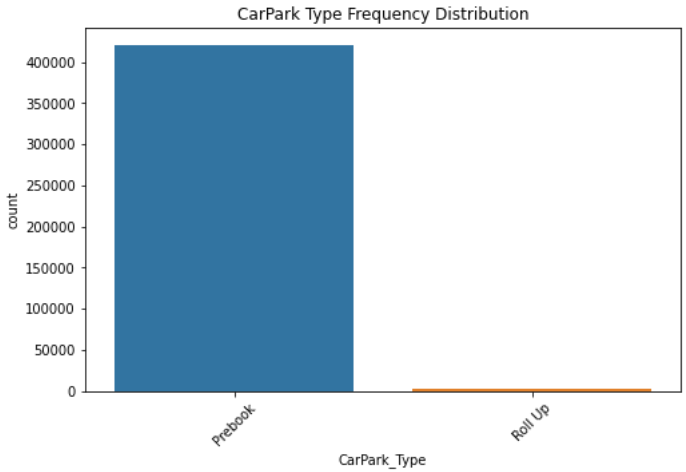
8,816 rows | Truncated data

Exploratory Data Analysis

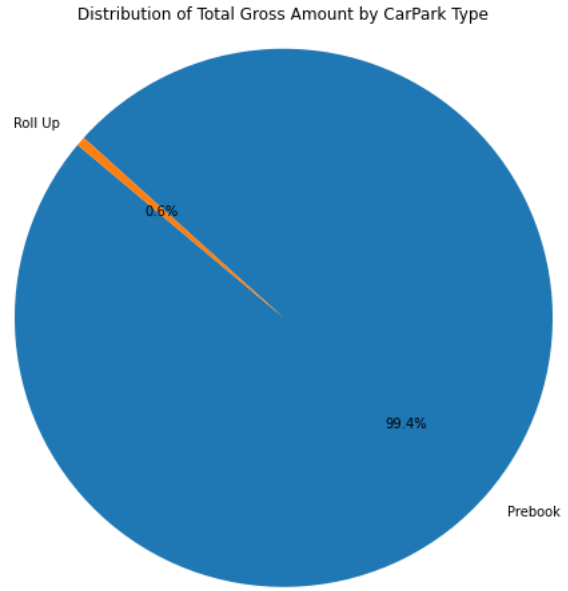
Period : 1st Jan 2021 - 3rd Oct. 2023

99.4% of the Gross Amount is earned from Pre-book CarPark

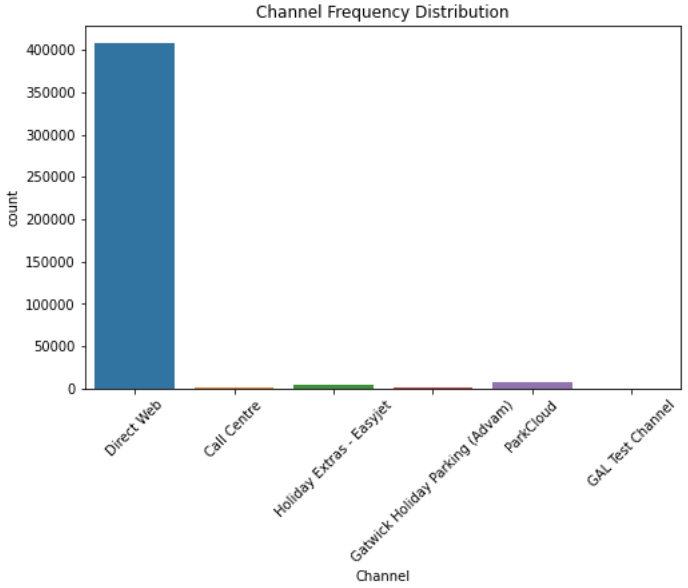
```
# Plot the frequency distribution of 'Parking_Type'
plt.figure(figsize=(8, 5))
sns.countplot(data=group_df, x='CarPark_Type')
plt.title('CarPark Type Frequency Distribution')
plt.xticks(rotation=45)
plt.show()
```



```
# Calculate the sum of Total_Gross_Amount for each CarPark_Type
carpark_type_totals = group_df.groupby('CarPark_Type')['Total_Gross_Amount'].sum()

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(carpark_type_totals, labels=carpark_type_totals.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Total Gross Amount by CarPark Type')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

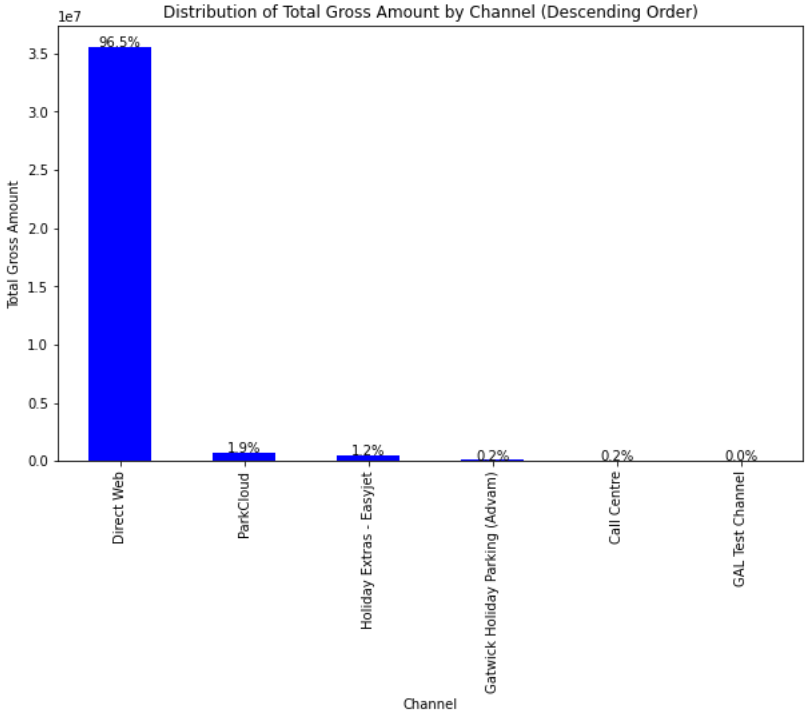



```
# Plot the frequency distribution of 'Channel'
plt.figure(figsize=(8, 5))
sns.countplot(data=group_df, x='Channel')
plt.title('Channel Frequency Distribution')
plt.xticks(rotation=45)
plt.show()
```

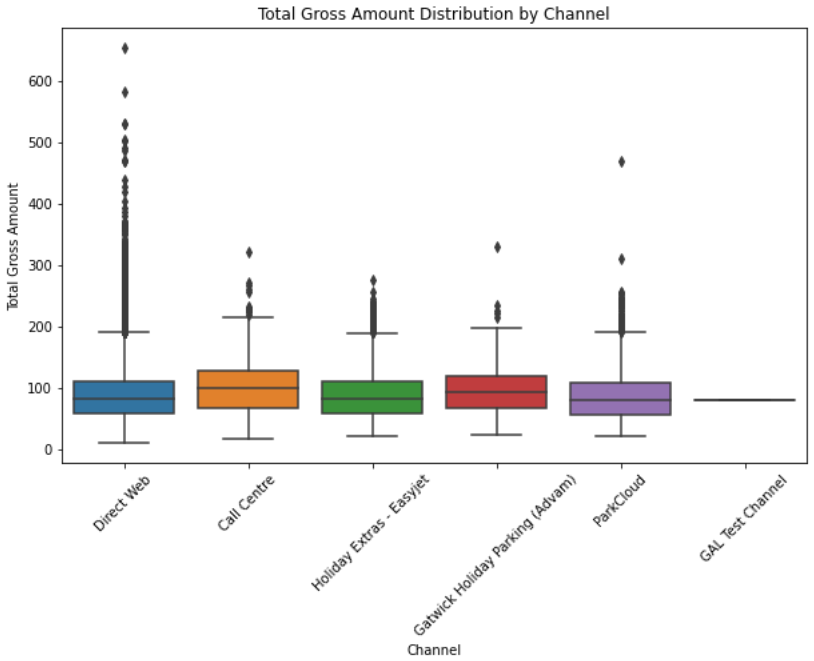


Direct Channel generates ~97% of the Total Gross Amount

```
# Calculate the sum of Total_Gross_Amount for each Channel
channel_totals = group_df.groupby('Channel')['Total_Gross_Amount'].sum()

# Calculate the percentages
total_sum = channel_totals.sum()
channel_percentages = (channel_totals / total_sum) * 100

# Sort the data in descending order based on percentages
sorted_channel_totals = channel_totals[channel_percentages.sort_values(ascending=False).index]

# Create a bar graph with the sorted data
plt.figure(figsize=(10, 6))
sorted_channel_totals.plot(kind='bar', color='blue')
plt.title('Distribution of Total Gross Amount by Channel (Descending Order)')
plt.ylabel('Total Gross Amount')
plt.xlabel('Channel')

# Display percentages on the bars
for i, v in enumerate(sorted_channel_totals):
    percentage = (v / total_sum) * 100
    plt.text(i, v + 5, f'{percentage:.1f}%', horizontalalignment='center')

plt.show()
```

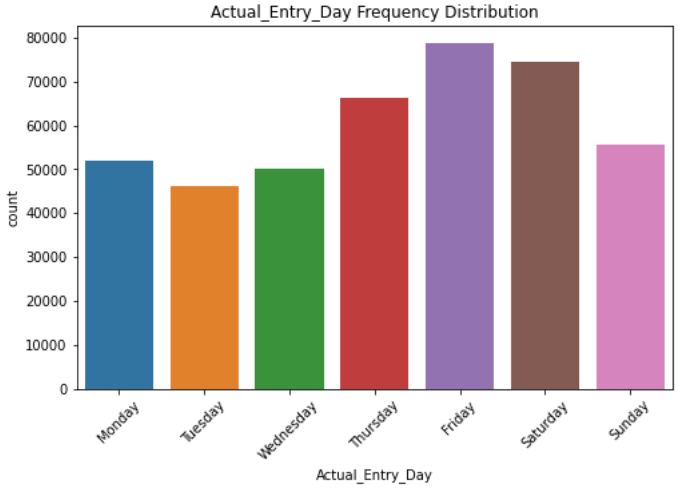



```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a box plot to show the distribution of Total_Gross_Amount for each channel
plt.figure(figsize=(10, 6))
sns.boxplot(data=group_df, x='Channel', y='Total_Gross_Amount')
plt.title('Total Gross Amount Distribution by Channel')
plt.xticks(rotation=45)
plt.ylabel('Total Gross Amount')
plt.xlabel('Channel')
plt.show()
```

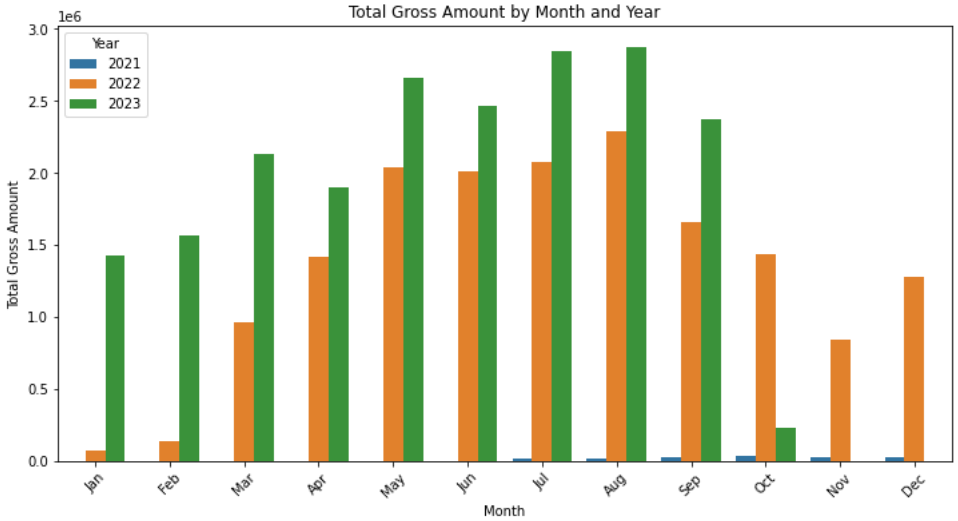


As expected Friday is the most booked Car Parks

```
# Define the custom order of days of the week
custom_day_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# Plot the frequency distribution of 'Actual_Entry_Day' with the custom order
plt.figure(figsize=(8, 5))
sns.countplot(data=group_df, x='Actual_Entry_Day', order=custom_day_order)
plt.title('Actual_Entry_Day Frequency Distribution')
plt.xticks(rotation=45)
plt.show()
```

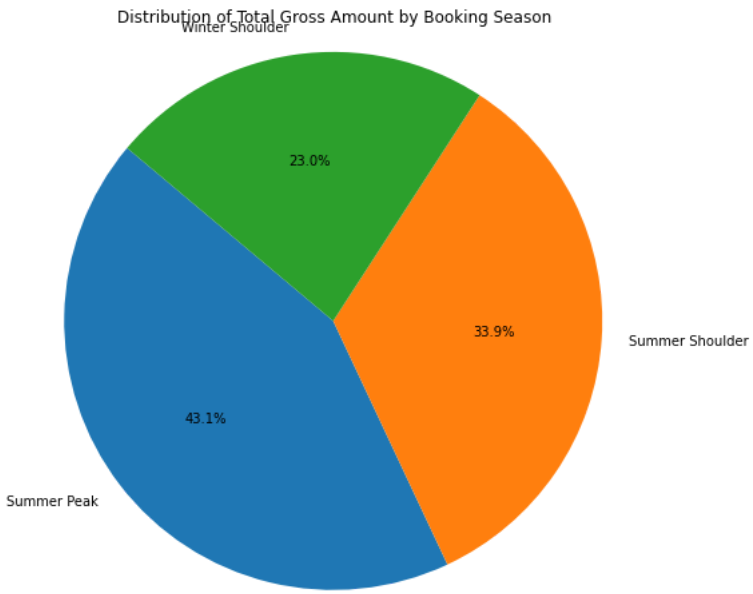



```
# Define the custom order of month names
custom_month_order = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

# Convert 'Booking_Date' to datetime format
group_df['Booking_Date'] = pd.to_datetime(group_df['Booking_Date'])

# Group by 'Year' and 'Month_Name' and aggregate the data
total_gross_amount_by_month_year = group_df.groupby(['Year', 'Month_Name'])['Total_Gross_Amount'].sum().reset_index()

# Create a bar graph to display Total Gross Amount by month and year
plt.figure(figsize=(12, 6))
sns.barplot(data=total_gross_amount_by_month_year, x='Month_Name', y='Total_Gross_Amount', hue='Year', order=custom_month_order)
plt.title('Total Gross Amount by Month and Year')
plt.xlabel('Month')
plt.ylabel('Total Gross Amount')
plt.xticks(rotation=45)
plt.legend(title='Year')
plt.show()
```



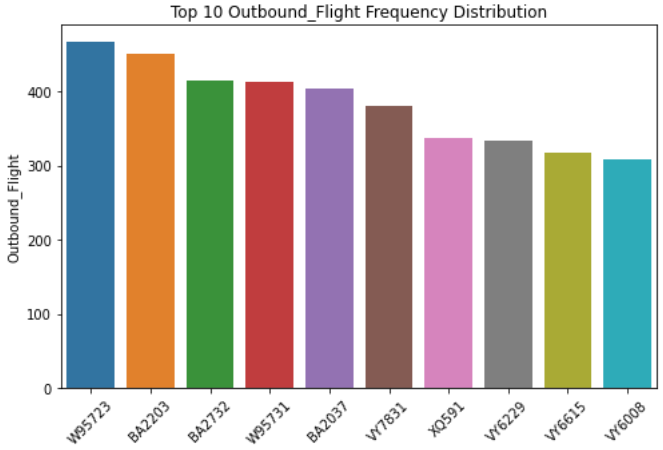
Summer season generates ~77% of the Total Gross Amount

```
# Calculate the sum of Total_Gross_Amount for each CarPark_Type
carpark_type_totals = group_df.groupby('Booking_Season')['Total_Gross_Amount'].sum()

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(carpark_type_totals, labels=carpark_type_totals.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Total Gross Amount by Booking Season')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



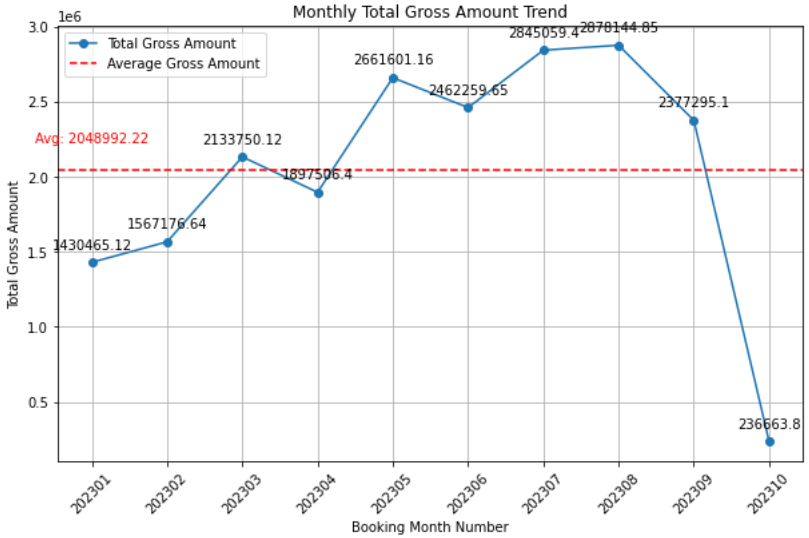
Customers who book the Car Parks usually takes the below Flights

```
# Filter the DataFrame to exclude "null" values and select the top 10 Outbound_Flight values
filtered_df = group_df[group_df['Outbound_Flight'] != "null"]
top_10_outbound_flight = filtered_df['Outbound_Flight'].value_counts().head(10)

# Plot the frequency distribution of the top 10 Outbound_Flight values
plt.figure(figsize=(8, 5))
sns.barplot(x=top_10_outbound_flight.index, y=top_10_outbound_flight)
plt.title('Top 10 Outbound_Flight Frequency Distribution')
plt.xticks(rotation=45)
plt.show()
```

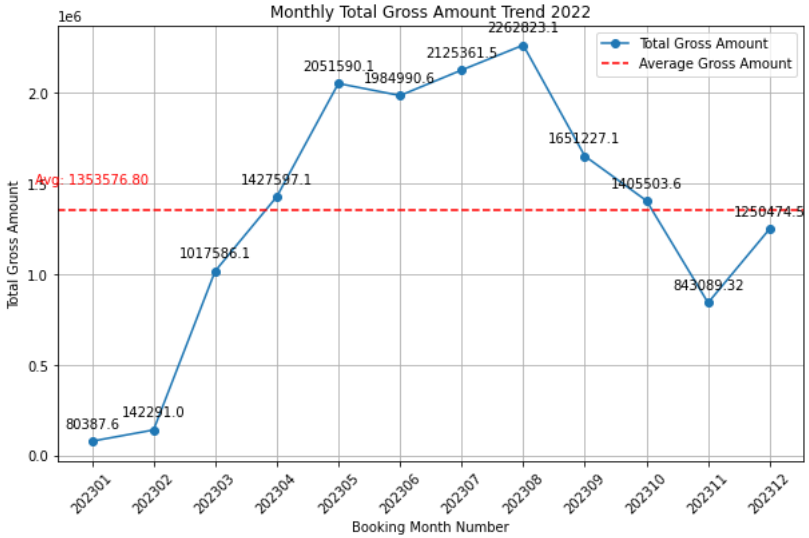


ST Long Stay Car park generated Avg. Gross Amount per month -- 2023 : 2.04M GBP | 2022 : 1.35M GBP | 2021 : 14.5K GBP per month

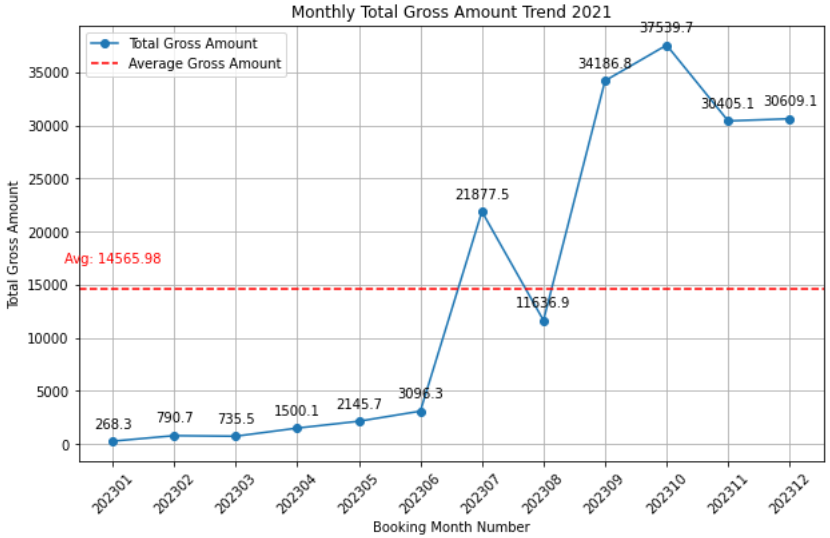
```
# Filter the DataFrame for the year 2023
group_df_2023 = group_df[group_df['Year'] == 2023]

# Group the data by 'Booking_Month_No' and calculate the total gross amount for each month
monthly_trend = group_df_2023.groupby('Booking_Month_No')['Total_Gross_Amount'].sum()

# Sort the data by 'Booking_Month_No' to ensure it's in chronological order
monthly_trend = monthly_trend.sort_index()

# Calculate the average Gross_Amount
average_gross_amount = monthly_trend.mean()

# Plot the monthly trend as a line chart with values on the markers
plt.figure(figsize=(10, 6))
plt.plot(monthly_trend.index, monthly_trend.values, marker='o', linestyle='-', label='Total Gross Amount')

# Annotate the values on the markers
for x, y in zip(monthly_trend.index, monthly_trend.values):
    plt.annotate(str(y), (x, y), textcoords="offset points", xytext=(0, 10), ha='center')

# Add a dotted line for the average Gross_Amount
plt.axhline(y=average_gross_amount, color='r', linestyle='--', label='Average Gross Amount')

# Annotate the average value on the dotted line
plt.annotate(f'Avg: {average_gross_amount:.2f}', xy=(monthly_trend.index[0], average_gross_amount), xytext=(0, 20),
             textcoords='offset points', color='r', ha='center')

plt.title('Monthly Total Gross Amount Trend')
plt.xlabel('Booking Month Number')
plt.ylabel('Total Gross Amount')
plt.xticks(rotation=45)
plt.grid()
plt.legend()
plt.show()
```


Monthly Total Gross Amount Trend
1e6
Total Gross Amount
Average Gross Amount
Avg: 2048992.22
1430465.12
1567176.64
2133750.12
1897506.4
2661601.16
2462259.65
2845059.4
2878144.85
2377295.1
236663.8
Total Gross Amount
Booking Month Number
202301
202302
202303
202304
202305
202306
202307
202308
202309
202310

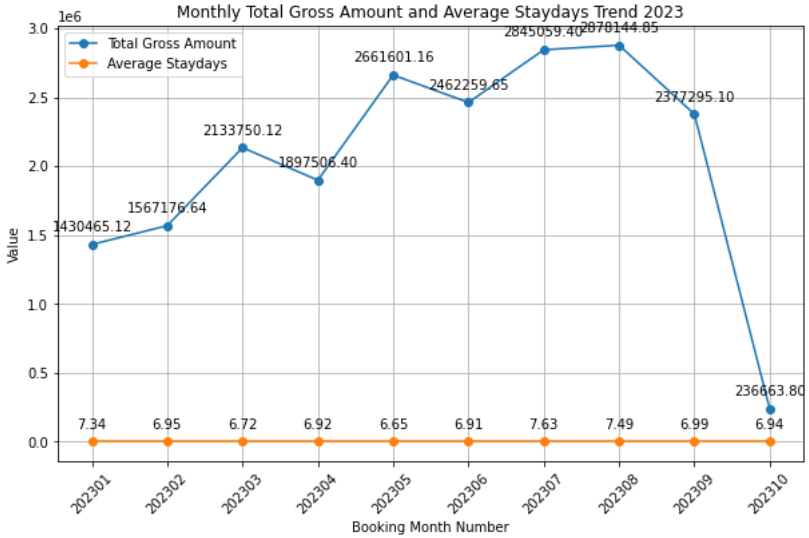
```
# Filter the DataFrame for the year 2022
group_df_2022 = group_df[group_df['Year'] == 2022]

# Group the data by 'Booking_Month_No' and calculate the total gross amount for each month
monthly_trend = group_df_2022.groupby('Booking_Month_No')['Total_Gross_Amount'].sum()

# Sort the data by 'Booking_Month_No' to ensure it's in chronological order
monthly_trend = monthly_trend.sort_index()

# Calculate the average Gross_Amount
average_gross_amount = monthly_trend.mean()

# Plot the monthly trend as a line chart with values on the markers
plt.figure(figsize=(10, 6))
plt.plot(monthly_trend.index, monthly_trend.values, marker='o', linestyle='-', label='Total Gross Amount')

# Annotate the values on the markers
for x, y in zip(monthly_trend.index, monthly_trend.values):
    plt.annotate(str(y), (x, y), textcoords="offset points", xytext=(0, 10), ha='center')

# Add a dotted line for the average Gross_Amount
plt.axhline(y=average_gross_amount, color='r', linestyle='--', label='Average Gross Amount')

# Annotate the average value on the dotted line
plt.annotate(f'Avg: {average_gross_amount:.2f}', xy=(monthly_trend.index[0], average_gross_amount), xytext=(0, 20),
             textcoords='offset points', color='r', ha='center')

plt.title('Monthly Total Gross Amount Trend 2022')
plt.xlabel('Booking Month Number')
plt.ylabel('Total Gross Amount')
plt.xticks(rotation=45)
plt.grid()
plt.legend()
plt.show()
```


Monthly Total Gross Amount Trend 2022
1e6
Total Gross Amount
Average Gross Amount
Avg: 1353576.80
80387.6
142291.0
1017586.1
1427597.1
2051590.1
1984990.6
2125361.5
2262823.1
1651227.1
1405503.6
843089.32
1250474.5
Total Gross Amount
Booking Month Number
202301
202302
202303
202304
202305
202306
202307
202308
202309
202310
202311
202312

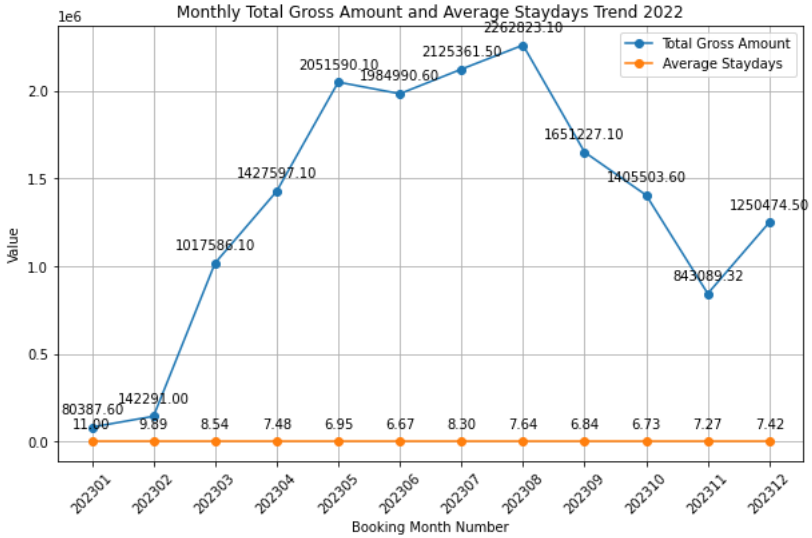
```
# Filter the DataFrame for the year 2021
group_df_2021 = group_df[group_df['Year'] == 2021]

# Group the data by 'Booking_Month_No' and calculate the total gross amount for each month
monthly_trend = group_df_2021.groupby('Booking_Month_No')['Total_Gross_Amount'].sum()

# Sort the data by 'Booking_Month_No' to ensure it's in chronological order
monthly_trend = monthly_trend.sort_index()

# Calculate the average Gross_Amount
average_gross_amount = monthly_trend.mean()

# Plot the monthly trend as a line chart with values on the markers
plt.figure(figsize=(10, 6))
plt.plot(monthly_trend.index, monthly_trend.values, marker='o', linestyle='-', label='Total Gross Amount')

# Annotate the values on the markers
for x, y in zip(monthly_trend.index, monthly_trend.values):
    plt.annotate(str(y), (x, y), textcoords="offset points", xytext=(0, 10), ha='center')

# Add a dotted line for the average Gross_Amount
plt.axhline(y=average_gross_amount, color='r', linestyle='--', label='Average Gross Amount')

# Annotate the average value on the dotted line
plt.annotate(f'Avg: {average_gross_amount:.2f}', xy=(monthly_trend.index[0], average_gross_amount), xytext=(0, 20),
             textcoords='offset points', color='r', ha='center')

plt.title('Monthly Total Gross Amount Trend 2021')
plt.xlabel('Booking Month Number')
plt.ylabel('Total Gross Amount')
plt.xticks(rotation=45)
plt.grid()
plt.legend()
plt.show()
```


Monthly Total Gross Amount Trend 2021
Total Gross Amount
Average Gross Amount
Avg: 14565.98
Total Gross Amount
Booking Month Number
268.3
790.7
735.5
1500.1
2145.7
3096.3
21877.5
11636.9
34186.8
37539.7
30405.1
30609.1
202301
202302
202303
202304
202305
202306
202307
202308
202309
202310
202311
202312

```
# Filter the DataFrame for the year 2023
group_df_2023 = group_df[group_df['Year'] == 2023]

# Calculate the total Gross Amount per month
monthly_gross_amount = group_df_2023.groupby('Booking_Month_No')['Total_Gross_Amount'].sum()

# Calculate the average Total Staydays per month
monthly_avg_staydays = group_df_2023.groupby('Booking_Month_No')['Total_Staydays'].mean()

# Sort the data by 'Booking_Month_No' to ensure it's in chronological order
monthly_gross_amount = monthly_gross_amount.sort_index()
monthly_avg_staydays = monthly_avg_staydays.sort_index()

# Plot the monthly trend as a line chart with values on the markers
plt.figure(figsize=(10, 6))
plt.plot(monthly_gross_amount.index, monthly_gross_amount.values, marker='o', linestyle='-', label='Total Gross Amount')
plt.plot(monthly_avg_staydays.index, monthly_avg_staydays.values, marker='o', linestyle='-', label='Average Staydays')

# Annotate the values on the markers for Total Gross Amount
for x, y in zip(monthly_gross_amount.index, monthly_gross_amount.values):
    plt.annotate(f'{y:.2f}', (x, y), textcoords="offset points", xytext=(0, 10), ha='center')

# Annotate the values on the markers for Average Staydays
for x, y in zip(monthly_avg_staydays.index, monthly_avg_staydays.values):
    plt.annotate(f'{y:.2f}', (x, y), textcoords="offset points", xytext=(0, 10), ha='center')

plt.title('Monthly Total Gross Amount and Average Staydays Trend 2023')
plt.xlabel('Booking Month Number')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.grid()
plt.legend()
plt.show()
```


Monthly Total Gross Amount and Average Staydays Trend 2023
1e6
Total Gross Amount
Average Staydays
Value
Booking Month Number
1430465.12
1567176.64
2133750.12
1897506.40
2661601.16
2462259.65
2845059.40
2870144.65
2377295.10
236663.80
7.34
6.95
6.72
6.92
6.65
6.91
7.63
7.49
6.99
6.94
202301
202302
202303
202304
202305
202306
202307
202308
202309
202310

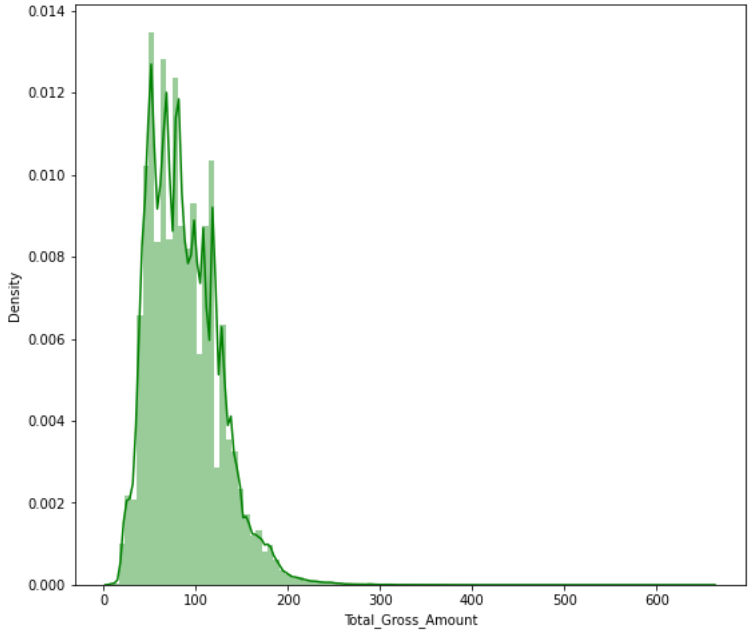
```
# Filter the DataFrame for the year 2022
group_df_2022 = group_df[group_df['Year'] == 2022]

# Calculate the total Gross Amount per month
monthly_gross_amount = group_df_2022.groupby('Booking_Month_No')['Total_Gross_Amount'].sum()

# Calculate the average Total Staydays per month
monthly_avg_staydays = group_df_2022.groupby('Booking_Month_No')['Total_Staydays'].mean()

# Sort the data by 'Booking_Month_No' to ensure it's in chronological order
monthly_gross_amount = monthly_gross_amount.sort_index()
monthly_avg_staydays = monthly_avg_staydays.sort_index()

# Plot the monthly trend as a line chart with values on the markers
plt.figure(figsize=(10, 6))
plt.plot(monthly_gross_amount.index, monthly_gross_amount.values, marker='o', linestyle='-', label='Total Gross Amount')
plt.plot(monthly_avg_staydays.index, monthly_avg_staydays.values, marker='o', linestyle='-', label='Average Staydays')

# Annotate the values on the markers for Total Gross Amount
for x, y in zip(monthly_gross_amount.index, monthly_gross_amount.values):
    plt.annotate(f'{y:.2f}', (x, y), textcoords="offset points", xytext=(0, 10), ha='center')

# Annotate the values on the markers for Average Staydays
for x, y in zip(monthly_avg_staydays.index, monthly_avg_staydays.values):
    plt.annotate(f'{y:.2f}', (x, y), textcoords="offset points", xytext=(0, 10), ha='center')

plt.title('Monthly Total Gross Amount and Average Staydays Trend 2022')
plt.xlabel('Booking Month Number')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.grid()
plt.legend()
plt.show()
```



Avg. ~89 GBP is generated per Booking for 2023

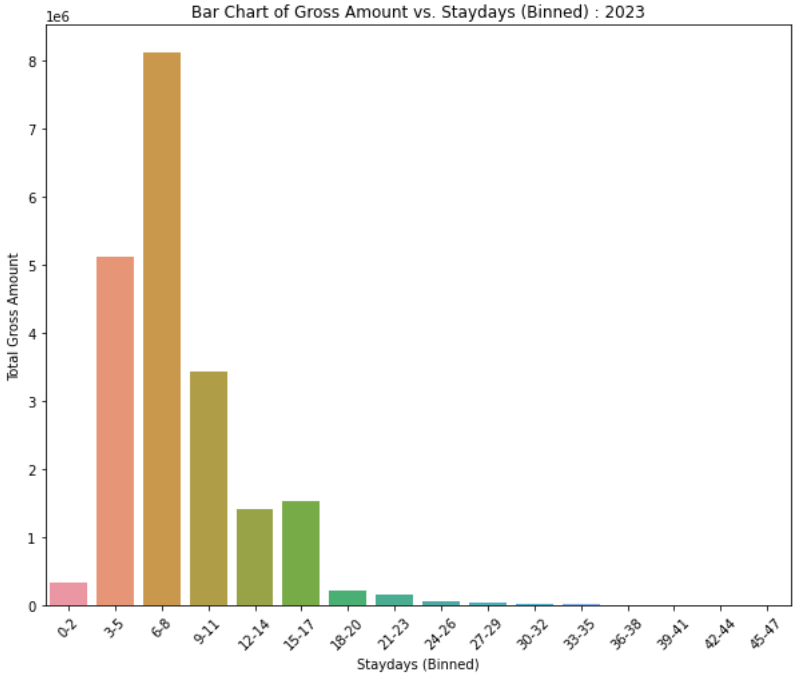
```
# Filter the DataFrame for the year 2023
group_df_2023 = group_df[group_df['Year'] == 2023]

print(group_df_2023['Total_Gross_Amount'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(group_df['Total_Gross_Amount'], color='g', bins=100, hist_kws={'alpha': 0.4});
```

```
count    229041.000000
mean         89.459626
std          37.783952
min          10.000000
25%          60.300000
50%          83.000000
75%         119.000000
max         655.000000
Name: Total_Gross_Amount, dtype: float64
```



Highest Gross Amount is earned when the CarPark is booked between 6-8 Stay Days (2023)

```
# Filter the DataFrame for the year 2023
group_df_2023 = group_df[group_df['Year'] == 2023]

# Create bins for Staydays with a size of 3
bin_edges = range(0, 51, 3)  # Adjust the number of edges
group_df_2023['Staydays_Bin'] = pd.cut(group_df_2023['Total_Staydays'], bins=bin_edges, right=False)

# Manually format the bin labels
bin_labels = [f"{i}-{i+2}" for i in range(0, 48, 3)]  # Adjust the number of labels

# Apply the custom bin labels
group_df_2023['Staydays_Bin'] = pd.cut(group_df_2023['Total_Staydays'], bins=bin_edges, labels=bin_labels, right=False)

# Create a bar chart to show the distribution of Gross Amount over binned Staydays
plt.figure(figsize=(10, 8))
sns.barplot(x="Staydays_Bin", y="Total_Gross_Amount", data=group_df_2023, estimator=sum, ci=None)
plt.title("Bar Chart of Gross Amount vs. Staydays (Binned) : 2023")
plt.xlabel("Staydays (Binned)")
plt.ylabel("Total Gross Amount")
plt.xticks(rotation=45)
plt.show()
```



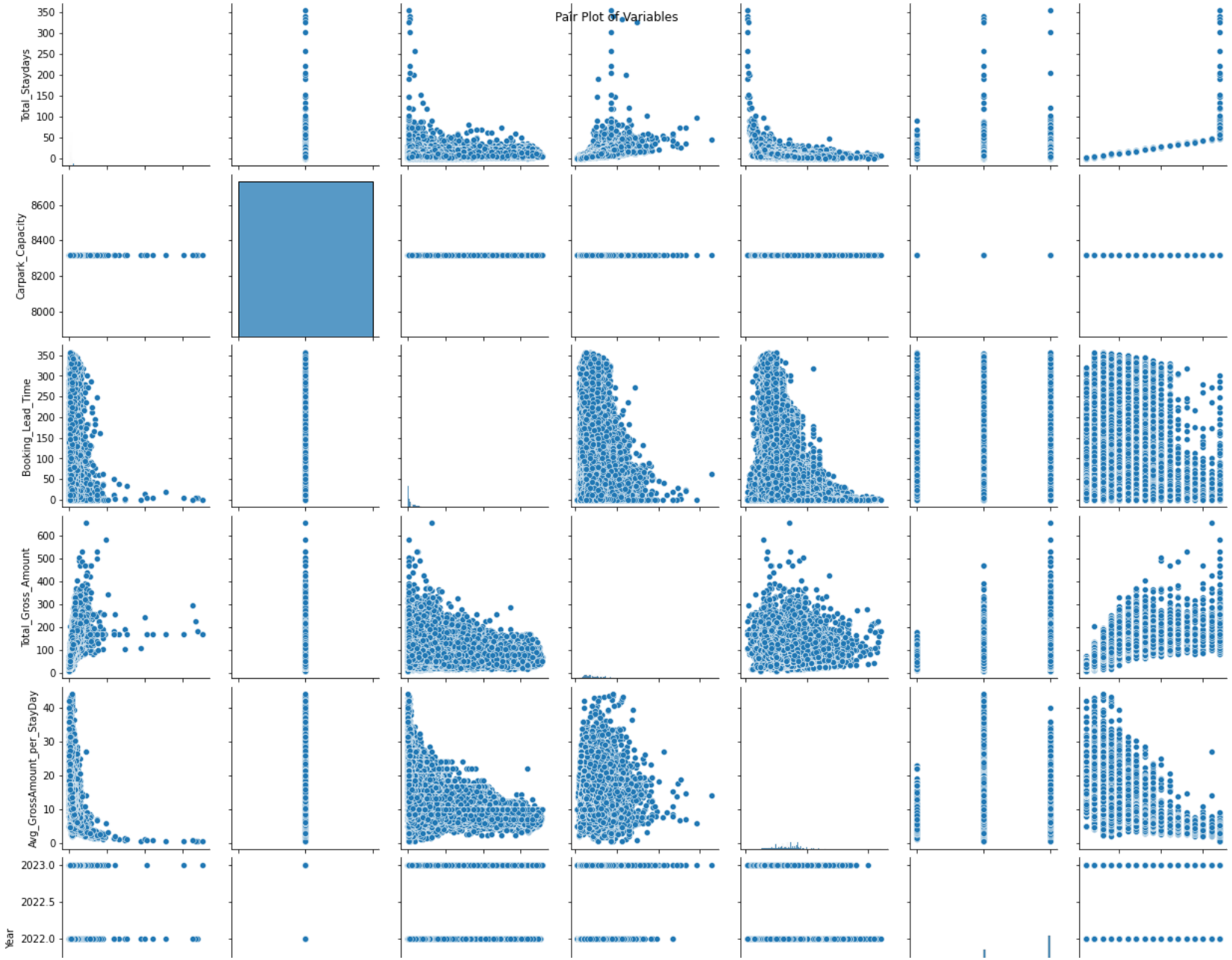
```
# Create bins for Staydays with a size of 3 in the Pandas DataFrame
bin_edges = range(0, 51, 3)  # Adjust the number of edges
group_df['Staydays_Bin'] = pd.cut(group_df['Total_Staydays'], bins=bin_edges, right=False)

# Convert 'Staydays_Bin' to a string
group_df['Staydays_Bin'] = group_df['Staydays_Bin'].astype(str)

# Create a new variable Staydays_Bin_No and assign values
group_df['Staydays_Bin_No'] = pd.cut(group_df['Total_Staydays'], bins=bin_edges, right=False, labels=False) + 1


# Replace 'Staydays_Bin' with "nan" with the value 49
group_df['Staydays_Bin'] = group_df['Staydays_Bin'].replace("nan", "49")

# Assign 'Staydays_Bin_No' to 17 where 'Staydays_Bin' is "49"
group_df.loc[group_df['Staydays_Bin'] == "49", 'Staydays_Bin_No'] = 17# Manually format the bin labels
bin_labels = [f"{i}-{i+2}" for i in range(0, 48, 3)]  # Adjust the number of labels

# Apply the custom bin labels
group_df_2023['Staydays_Bin'] = pd.cut(group_df_2023['Total_Staydays'], bins=bin_edges, labels=bin_labels, right=False)



# Create a new column 'Staydays_Bin_Grouped' based on 'Staydays_Bin_No' with grouping from 8 to 17
group_df['Staydays_Bin_Grouped'] = group_df['Staydays_Bin_No'].apply(lambda x: '9' if 8 <= x <= 17 else str(x))

# Convert 'Staydays_Bin_Grouped' to integers
#group_df['Staydays_Bin_Grouped'] = group_df['Staydays_Bin_Grouped'].apply(lambda x: int(x) if x.isdigit() else x)

# Print the updated DataFrame
print(group_df[['Staydays_Bin_No', 'Staydays_Bin_Grouped']])
```

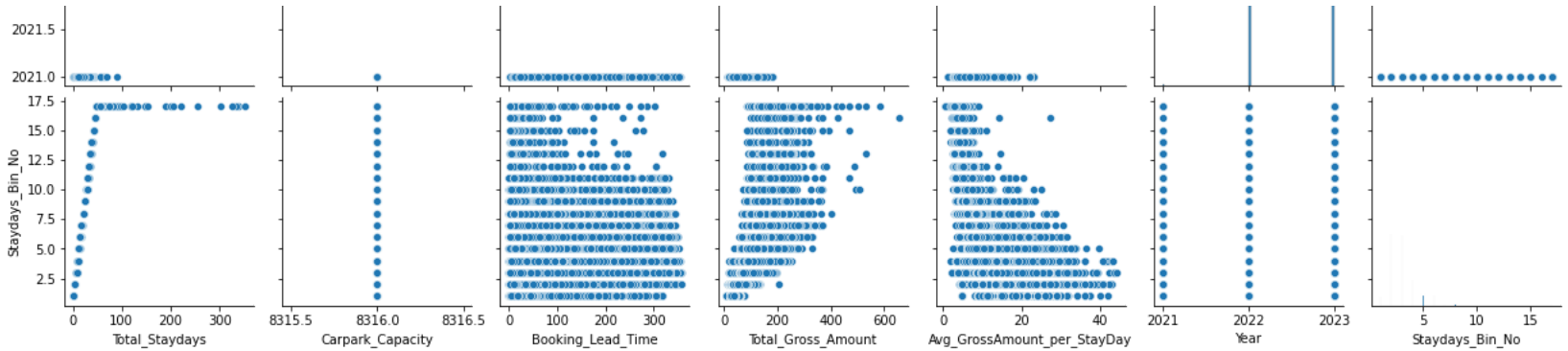
```
       Staydays_Bin_No Staydays_Bin_Grouped
0                  3.0                  3.0
1                  6.0                  6.0
2                  6.0                  6.0
3                  6.0                  6.0
4                  6.0                  6.0
...                ...                  ...
422945             2.0                  2.0
422946             3.0                  3.0
422947             2.0                  2.0
422948             5.0                  5.0
422949             2.0                  2.0

[422950 rows x 2 columns]
```

```
group_df.describe()
```

	Total_Staydays	Carpark_Capacity	Booking_Lead_Time	Total_Gross_Amount	Avg_GrossAmount_per_StayDay	Year	Staydays_Bin_No
count	422950.000000	422950.0	422950.000000	422950.000000	422950.000000	422950.000000	422950.000000
mean	7.241568	8316.0	27.431289	87.262408	13.245281	2022.535252	3.027798
std	4.318966	0.0	44.774042	37.340297	3.633934	0.511192	1.369982
min	1.000000	8316.0	0.000000	10.000000	0.484419	2021.000000	1.000000
25%	4.000000	8316.0	3.000000	58.250000	10.800000	2022.000000	2.000000
50%	7.000000	8316.0	11.000000	81.000000	13.111111	2023.000000	3.000000
75%	8.000000	8316.0	31.000000	111.000000	15.637500	2023.000000	3.000000
max	353.000000	8316.0	357.000000	655.000000	44.200000	2023.000000	17.000000

```
# Pair plot to visualize multiple variables
sns.pairplot(group_df, kind="scatter")
plt.suptitle("Pair Plot of Variables")
plt.show()
```


Pair Plot of Variables
Total_Staydays
Carpark_Capacity
Booking_Lead_Time
Total_Gross_Amount
Avg_GrossAmount_per_StayDay
Year

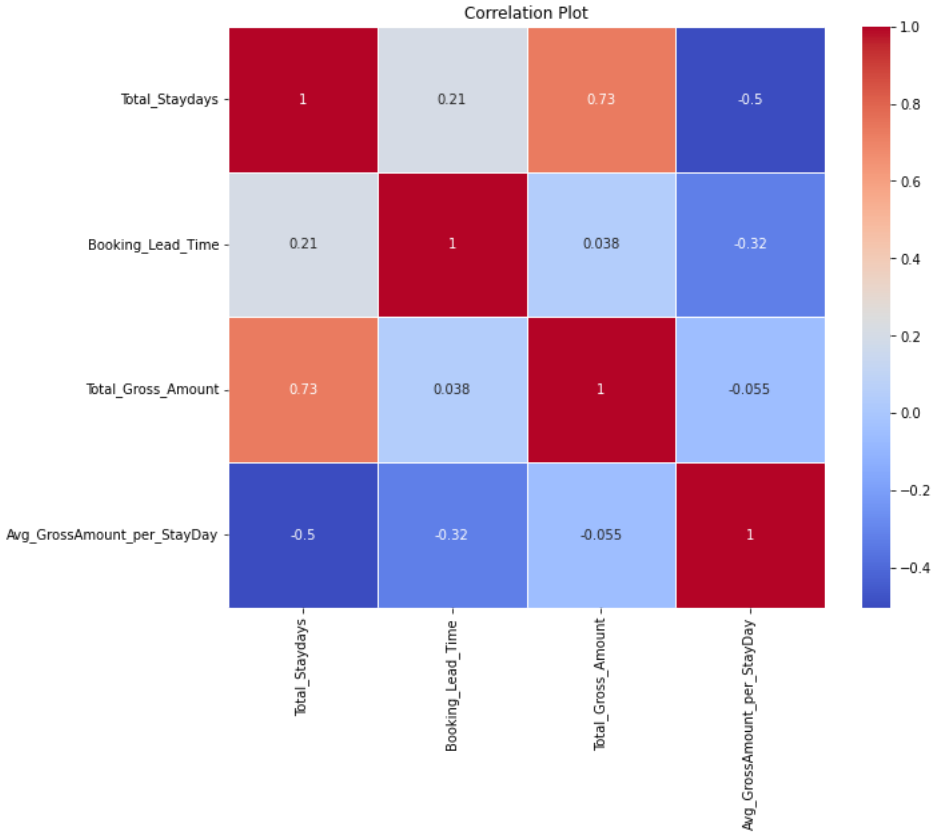


There is a strong correlation between Stay Days & Gross Amount ~0.77

```
# Select the relevant columns for the correlation matrix
correlation_columns = ['Total_Staydays', 'Staydays_Bin', 'Booking_Lead_Time', 'Total_Gross_Amount', 'Avg_GrossAmount_per_StayDay']

# Create a subset of the DataFrame with the selected columns
subset_df = group_df[correlation_columns]

# Calculate the correlation matrix
correlation_matrix = subset_df.corr()

# Display the correlation matrix:
correlation_matrix
```

	Total_Staydays	Booking_Lead_Time	Total_Gross_Amount	Avg_GrossAmount_per_StayDay
Total_Staydays	1.000000	0.206661	0.726134	-0.503918
Booking_Lead_Time	0.206661	1.000000	0.037519	-0.322541
Total_Gross_Amount	0.726134	0.037519	1.000000	-0.055005
Avg_GrossAmount_per_StayDay	-0.503918	-0.322541	-0.055005	1.000000

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
correlation_matrix = subset_df.corr()

# Create a heatmap for the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title('Correlation Plot')
plt.show()
```



Model Build : Feature Engineering

Create Dummy variables for Booking Season, Channel and Staydays_Bin

```
print(group_df.columns)
```

```
Index(['booking_summary_source_booking_num', 'CarPark_Type', 'Booking_Date',
       'Entry_Date', 'Exit_Date', 'Booking_Season', 'Entry_Season',
       'Exit_Season', 'Channel', 'Actual_Entry_Day', 'Actual_Exit_Day',
       'Booking_Week', 'Booking_Month_No', 'Booking_Entry_Month_No',
       'Booking_Exit_Month_No', 'Outbound_Flight', 'Total_Staydays',
       'Carpark_Capacity', 'Booking_Lead_Time', 'Total_Gross_Amount',
       'Avg_GrossAmount_per_StayDay', 'Month_Name', 'Year', 'Staydays_Bin',
       'Staydays_Bin_No', 'Staydays_Bin_Grouped'],
      dtype='object')
```

```
# Create dummy (binary) variables for 'Booking_Season'
booking_season_dummies = pd.get_dummies(group_df['Booking_Season'], prefix='Booking_Season')

# Create dummy (binary) variables for 'Staydays_Bin_No'
staydays_bin_grouped_dummies = pd.get_dummies(group_df['Staydays_Bin_Grouped'], prefix='Staydays_Bin_Grouped')

# Concatenate the dummy variables with the original DataFrame
group_df = pd.concat([group_df, booking_season_dummies, staydays_bin_grouped_dummies], axis=1)

# Drop the original columns for which dummy variables were created
#group_df.drop(['Booking_Season', 'Staydays_Bin_Grouped'], axis=1, inplace=True)
```

```
group_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 422950 entries, 0 to 422949
Data columns (total 37 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   booking_summary_source_booking_num  422950 non-null  object
 1   CarPark_Type                       422950 non-null  object
 2   Booking_Date                       422950 non-null  datetime64[ns]
 3   Entry_Date                         422950 non-null  object
 4   Exit_Date                          422950 non-null  object
 5   Booking_Season                     422950 non-null  object
 6   Entry_Season                       422950 non-null  object
 7   Exit_Season                        422950 non-null  object
 8   Channel                            422950 non-null  object
 9   Actual_Entry_Day                   422950 non-null  object
 10  Actual_Exit_Day                    422950 non-null  object
 11  Booking_Week                       422950 non-null  object
 12  Booking_Month_No                   422950 non-null  object
```

```
13  Booking_Entry_Month_No              422950 non-null  object
14  Booking_Exit_Month_No               422950 non-null  object
15  Outbound Flight                     39091 non-null   object
```

```
from sklearn.preprocessing import LabelEncoder

# Create a label encoder
label_encoder = LabelEncoder()

# Apply label encoding to 'booking_summary_source_booking_num'
group_df['Booking_Num_Encoded'] = label_encoder.fit_transform(group_df['booking_summary_source_booking_num'])

# Now 'Booking_Num_Encoded' contains the numerical representation of 'booking_summary_source_booking_num'
```

Model Data with Numberic Columnns only

```
model_data = group_df.select_dtypes(include=[np.number])
display(model_data)
```

Table

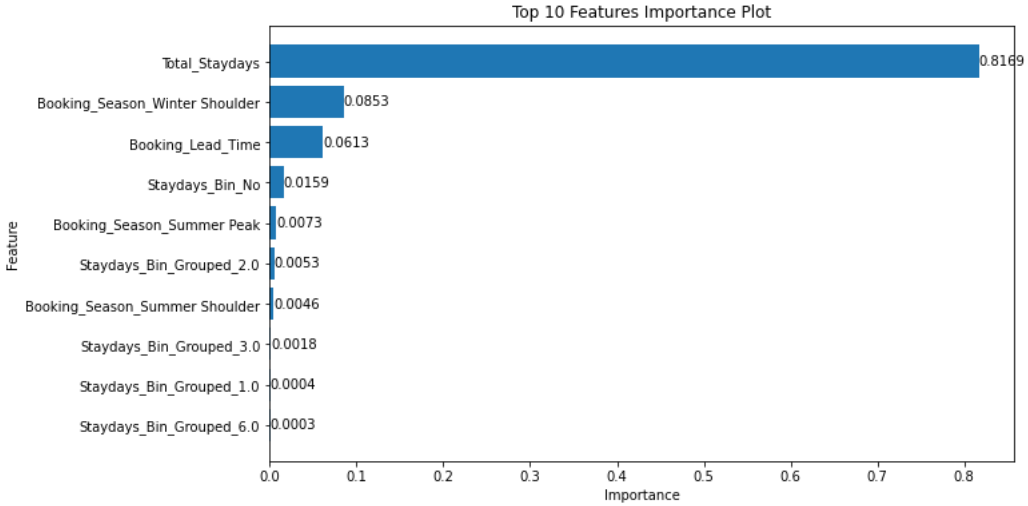
	Total_Staydays	Carpark_Capacity	Booking_Lead_Time	Total_Gross_Amount	Avg_GrossAmount_per_StayDay	Year	Staydays_Bin_No	Booking_Season_Summer Peak	Booking
1	8	8316	214	45.6	5.7	2021	3	0	0
2	15	8316	202	85	5.666666666666667	2021	6	0	0
3	15	8316	119	85	5.666666666666667	2021	6	0	0
4	15	8316	119	85	5.666666666666667	2021	6	0	0
5	16	8316	110	89	5.5625	2021	6	0	0
6	16	8316	220	89	5.5625	2021	6	0	0
7	4	8316	105	29	7.25	2021	2	0	0

10,000 rows | Truncated data

Feature Importance using Random Forest Model

Total Stay Days is the most significant variable that impacts the Gross Amount

```
# Select numeric features and target variable
X = model_data.select_dtypes(include=[np.number]).drop(columns=['Total_Gross_Amount','Avg_GrossAmount_per_StayDay','Booking_Num_Encoded', 'Year'])  # Numeric features
y = model_data['Total_Gross_Amount']  # Target variable

# Create and fit a Random Forest model
modelRF = RandomForestRegressor(random_state=42)
modelRF.fit(X, y)

# Get feature importances from the model
feature_importances = modelRF.feature_importances_

# Create a DataFrame to hold the feature names and their importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort the features by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Select the top 10 features with the highest importance
top_10_features = feature_importance_df.head(10)

# Create a bar plot to visualize feature importance with values displayed
plt.figure(figsize=(10, 6))
bars = plt.barh(top_10_features['Feature'], top_10_features['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Top 10 Features Importance Plot')

# Display values on the bars
for bar in bars:
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{bar.get_width():.4f}', va='center')

plt.gca().invert_yaxis()  # Invert the y-axis to show the most important features at the top
plt.show()
```



Model Build : Regression Models

```
print(model_data.columns)
```

```
Index(['Total_Staydays', 'Carpark_Capacity', 'Booking_Lead_Time',
       'Total_Gross_Amount', 'Avg_GrossAmount_per_StayDay', 'Year',
       'Staydays_Bin_No', 'Booking_Season_Summer Peak',
       'Booking_Season_Summer Shoulder', 'Booking_Season_Winter Shoulder',
       'Staydays_Bin_Grouped_1.0', 'Staydays_Bin_Grouped_2.0',
       'Staydays_Bin_Grouped_3.0', 'Staydays_Bin_Grouped_4.0',
       'Staydays_Bin_Grouped_5.0', 'Staydays_Bin_Grouped_6.0',
       'Staydays_Bin_Grouped_7.0', 'Staydays_Bin_Grouped_9',
       'Booking_Num_Encoded'],
      dtype='object')
```

Model performed poor when Avg_GrossAmount_per_StayDay is taken as Target variable

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
import statsmodels.api as sm

# Select features (X) and the target variable (Y)
X = model_data[['Total_Staydays', 'Booking_Lead_Time','Booking_Season_Summer Peak', 'Booking_Season_Summer Shoulder', 'Booking_Season_Winter Shoulder']]
y = model_data['Avg_GrossAmount_per_StayDay']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model with an intercept
model_PR = LinearRegression(fit_intercept=True)

# Fit the model on the training data
model_PR.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model_PR.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

# Calculate MAPE (Mean Absolute Percentage Error)
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
print("R-squared (R2):", r2)

# Fit an ordinary least squares (OLS) model
X_train_with_intercept = sm.add_constant(X_train)  # Add an intercept column to X_train
model_PR = sm.OLS(y_train, X_train_with_intercept).fit()

# Print the ANOVA table
print(model_PR.summary())
```

```
Mean Squared Error (MSE): 7.619171987004791
Mean Absolute Error (MAE): 2.03556909644048
Mean Absolute Percentage Error (MAPE): 16.76382191483038 %
R-squared (R2): 0.4247878068396399
                                OLS Regression Results
```

```
==============================================================================
Dep. Variable:     Avg_GrossAmount_per_StayDay   R-squared:                       0.415
Model:                                     OLS   Adj. R-squared:                  0.415
Method:                          Least Squares   F-statistic:                 5.998e+04
Date:                         Wed, 22 Nov 2023   Prob (F-statistic):               0.00
Time:                                 20:04:49   Log-Likelihood:            -8.2590e+05
No. Observations:                       338360   AIC:                          1.652e+06
Df Residuals:                           338355   BIC:                          1.652e+06
Df Model:                                    4
Covariance Type:                     nonrobust
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                         12.2344      0.007   1727.200      0.000      12.221      12.248
Total_Staydays                -0.3773      0.001   -334.358      0.000      -0.380      -0.375
Booking_Lead_Time             -0.0160      0.000   -146.044      0.000      -0.016      -0.016
```

Regression Model

Model performed with ~78% Accuracy (Jan 2021- Oct 2023) when Total Gross Amount is taken as Target variable. Model accuracy increased to ~82% for 2023.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
import statsmodels.api as sm

# Select features (X) and the target variable (Y)
X = model_data[['Total_Staydays', 'Booking_Lead_Time','Booking_Season_Summer Peak', 'Booking_Season_Summer Shoulder', 'Booking_Season_Winter Shoulder']]
y = model_data['Total_Gross_Amount']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model with an intercept
model = LinearRegression(fit_intercept=True)

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

# Calculate MAPE (Mean Absolute Percentage Error)
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
print("R-squared (R2):", r2)

# Fit an ordinary least squares (OLS) model on the test data
X_test_with_intercept = sm.add_constant(X_test)  # Add an intercept column to X_test
model_ols_test = sm.OLS(y_test, X_test_with_intercept).fit()

# Print the OLS summary for the test data
print(model_ols_test.summary())
```

```
Mean Squared Error (MSE): 525.3766735406256
Mean Absolute Error (MAE): 16.506022182444834
Mean Absolute Percentage Error (MAPE): 21.8052531986222 %
R-squared (R2): 0.6247072111620345
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:     Total_Gross_Amount   R-squared:                       0.626
Model:                            OLS   Adj. R-squared:                  0.626
Method:                 Least Squares   F-statistic:                 3.535e+04
Date:                Wed, 22 Nov 2023   Prob (F-statistic):               0.00
Time:                        20:04:55   Log-Likelihood:            -3.8486e+05
No. Observations:               84590   AIC:                          7.697e+05
Df Residuals:                   84585   BIC:                          7.698e+05
Df Model:                           4
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           29.9142      0.118    254.265      0.000      29.684      30.145
Total_Staydays                   6.7355      0.019    356.934      0.000       6.698       6.772
Booking_Lead_Time               -0.0839      0.002    -46.743      0.000      -0.087      -0.080
```

Insights:

1. For each additional day of stay, the Total Gross Amount is predicted to increase by avg. GBP 6.7

Above model with Actual and Predicted Gross Amount in a single DataFrame is shown below:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
import statsmodels.api as sm

# Select features (X) and the target variable (Y)
X = group_df[['Total_Staydays', 'Booking_Lead_Time','Booking_Season_Summer Peak', 'Booking_Season_Summer Shoulder', 'Booking_Season_Winter Shoulder']]
y = group_df['Total_Gross_Amount']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model with an intercept
model = LinearRegression(fit_intercept=True)

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

# Calculate MAPE (Mean Absolute Percentage Error)
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
print("R-squared (R2):", r2)

# Fit an ordinary least squares (OLS) model on the test data
X_test_with_intercept = sm.add_constant(X_test)  # Add an intercept column to X_test
model_ols_test = sm.OLS(y_test, X_test_with_intercept).fit()

# Print the OLS summary for the test data
print(model_ols_test.summary())
```

```
Mean Squared Error (MSE): 525.3766735406256
Mean Absolute Error (MAE): 16.506022182444834
Mean Absolute Percentage Error (MAPE): 21.8052531986222 %
R-squared (R2): 0.6247072111620345
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:     Total_Gross_Amount   R-squared:                       0.626
Model:                            OLS   Adj. R-squared:                  0.626
Method:                 Least Squares   F-statistic:                 3.535e+04
Date:                Wed, 22 Nov 2023   Prob (F-statistic):               0.00
Time:                        20:05:01   Log-Likelihood:            -3.8486e+05
No. Observations:               84590   AIC:                          7.697e+05
Df Residuals:                   84585   BIC:                          7.698e+05
Df Model:                           4
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           29.9142      0.118    254.265      0.000      29.684      30.145
Total_Staydays                   6.7355      0.019    356.934      0.000       6.698       6.772
Booking_Lead_Time               -0.0839      0.002    -46.743      0.000      -0.087      -0.080
```

```
# Create a DataFrame to combine X_test, y_test, and y_pred
combined_results = pd.DataFrame({'Total_Gross_Amount_Actual': y_test, 'Total_Gross_Amount_Predicted': y_pred})
```

```
# Create a DataFrame for y_pred
Total_Gross_Amount_Predicted = pd.DataFrame(y_pred)
```

XGBoost Model

XGB Model has ~86% Accuracy, but due to its inherit property it does not generate co-eff. values. As we require co-eff. values, we cannot use this model for the Price Prediction.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Your data preprocessing code goes here

# Step 1: Split the dataset into features (X) and target variable (Y)
X = model_data[['Total_Staydays', 'Booking_Lead_Time', 'Booking_Season_Summer Peak', 'Booking_Season_Summer Shoulder', 'Booking_Season_Winter Shoulder']]
Y = model_data['Total_Gross_Amount']

# Step 2: Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Step 3: Train the XGBoost model
xgb_model = XGBRegressor()
xgb_model.fit(X_train, Y_train)

# Step 4: Make predictions on the test data
Y_pred = xgb_model.predict(X_test)

# Step 5: Calculate regression metrics using the test data
mse = mean_squared_error(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

# Make predictions on the test data using the XGBoost model
Y_pred = xgb_model.predict(X_test)

# Calculate the Mean Absolute Percentage Error (MAPE)
mape = np.mean(np.abs((Y_test - Y_pred) / Y_test)) * 100

# Print the XGBoost model
print(xgb_model)

# Print the Evaluation metrics on Test data:
print("R-squared (R2) from Statsmodels:", r2)
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
```

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
             grow_policy='depthwise', importance_type=None,
             interaction_constraints='', learning_rate=0.300000012, max_bin=256,
             max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
             max_depth=6, max_leaves=0, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=0,
```

```
            num_parallel_tree=1, predictor='auto', random_state=0, ...)
R-squared (R2) from Statsmodels: 0.7943216273714936
Mean Absolute Percentage Error (MAPE): 13.958893048789287 %
```

```
  importance_scores = xgb_model.feature_importances_
  importance_scores
```

```
Out[58]: array([0.47087035, 0.01949556, 0.04642494, 0.01523762, 0.4479715 ],
      dtype=float32)
```

LightGBM Model

LGB also has ~86% Accuracy, but due to its inherit property it does not generate co-eff. values. As we require co-eff. values, we cannot use this model for the Price Prediction.

```
import pandas as pd
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

# Define predictor variables and target variable
X = model_data[['Total_Staydays', 'Booking_Lead_Time','Booking_Season_Summer Peak', 'Booking_Season_Summer Shoulder', 'Booking_Season_Winter Shoulder']]
y = model_data['Total_Gross_Amount']

# Split the data into training and testing sets (if needed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a LightGBM dataset
train_data = lgb.Dataset(X_train, label=y_train)

# Define model parameters
params = {
    'objective': 'regression',
    'metric': 'mse',
    'boosting_type': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9
}

# Train the model
modelGB = lgb.train(params, train_data, num_boost_round=100)

# Make predictions
y_pred = modelGB.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

# Calculate MAPE (Mean Absolute Percentage Error)
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape, "%")
print("R-squared (R2):", r2)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.002494 seconds.
```

```
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 328
[LightGBM] [Info] Number of data points in the train set: 338360, number of used features: 5
[LightGBM] [Info] Start training from score 87.261916
Mean Squared Error (MSE): 287.78705044724126
Mean Absolute Error (MAE): 12.015308060030687
Mean Absolute Percentage Error (MAPE): 14.087035449959817 %
R-squared (R2): 0.7944248190047482
```

Add Predicted data from Regression Model to Original data(2023)

```
# Create a new column in the model_data DataFrame to store the predictions
group_df['Predicted_Total_Gross_Amount'] = model.predict(X)

group_df_final_data = group_df

# Filter the DataFrame for the year 2023
group_df_final_data_2023 = group_df_final_data[group_df_final_data['Year'] == 2023]

# Remove values from "T" onwards in the Booking_Date column
group_df_final_data_2023['Booking_Date'] = group_df_final_data_2023['Booking_Date'].dt.strftime('%Y-%m-%d')

display(group_df_final_data_2023)
```

Table

	booking_summary_source_booking_num	CarPark_Type	Booking_Date	Entry_Date	Exit_Date	Booking_Season	Entry_Season	Exit_Season	Channel	Actual_Entry_Day
1	DW01499503NIC	Prebook	2023-01-11	2023-02-04	2023-02-15	Winter Shoulder	Winter Shoulder	Winter Shoulder	Direct Web	Saturday
2	DW01886262VIC	Roll Up	2023-06-03	2023-06-03	2023-06-13	Summer Shoulder	Summer Shoulder	Summer Shoulder	Direct Web	Sunday
3	DW01979702LAU	Roll Up	2023-02-11	2023-02-11	2023-02-17	Winter Shoulder	Winter Shoulder	Winter Shoulder	Direct Web	Thursday
4	DW02281761ALL	Roll Up	2023-03-17	2023-03-17	2023-03-28	Winter Shoulder	Winter Shoulder	Winter Shoulder	Direct Web	Monday
5	DW02377988JAS	Roll Up	2023-08-09	2023-08-09	2023-09-02	Summer Peak	Summer Peak	Summer Peak	Direct Web	Thursday
6	DW02379301KEL	Roll Up	2023-04-01	2023-04-01	2023-04-16	Summer Shoulder	Summer Shoulder	Summer Shoulder	Direct Web	Saturday
7	DW02385238ANT	Roll Up	2023-01-14	2023-01-14	2023-01-25	Winter Shoulder	Winter Shoulder	Winter Shoulder	Direct Web	Tuesday

6,825 rows | Truncated data

Save the above Output with Price Prediction for Dashboard Building

```
outname = 'group_df_final_data_2023.csv'
outdir = '/dbfs/FileStore/'
group_df_final_data.to_csv(outdir+outname, index=False, encoding="utf-8")
```

AUTOML

```
#Data used for AutoML
model_data
```

	Total_Staydays	Carpark_Capacity	Booking_Lead_Time	Total_Gross_Amount	Avg_GrossAmount_per_StayDay	Year	Staydays_Bin_No	Booking_Season_Summer Peak	Booking_Season_Summer Shoulder	Booking_Season_Winter Shoulder	Stayda
0	8	8316.0	214.0	45.6	5.700000	2021	3.0	0	0	1	
1	15	8316.0	202.0	85.0	5.666667	2021	6.0	0	0	1	
2	15	8316.0	119.0	85.0	5.666667	2021	6.0	0	0	1	
3	15	8316.0	119.0	85.0	5.666667	2021	6.0	0	0	1	
4	16	8316.0	110.0	89.0	5.562500	2021	6.0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...
422945	4	8316.0	1.0	67.0	16.750000	2023	2.0	1	0	0	
422946	8	8316.0	24.0	109.0	13.625000	2023	3.0	1	0	0	
422947	3	8316.0	1.0	52.0	17.333333	2023	2.0	1	0	0	
422948	12	8316.0	12.0	125.0	10.416667	2023	5.0	1	0	0	
422949	5	8316.0	11.0	80.0	16.000000	2023	2.0	0	0	1	

422950 rows × 19 columns