➤ SITUATION : The profitability of the airlines industry has come under considerable strain due to a range of factors. In light of this situation, it has become imperative to swiftly embrace a comprehensive and strategic approach, ultimately revitalizing profitability in the long run.

➤ TASK : Identify the opportunities for increasing the occupancy rate on flights that currently show lower performance metrics. This strategic initiative carries the potential to result in a significant boost to the overall profitability of the airline.

➤ ACTION : After gathering the necessary data, we utilized sophisticated analytical tools to strategically uncover intricate patterns and correlations contributing to low flight occupancy. These tools played a pivotal role in revealing valuable insights crucial for a comprehensive understanding of the underlying dynamics in the airline's data landscape.

➤ RESULT : With proactive measures to address low occupancy rates, the airline industry seeks to amplify revenue potential. By skillfully implementing proposed strategies and closely monitoring their impact, the industry targets specific reductions in occupancy rates and subsequent revenue increases.

# Importing Libraries

In [28]:
```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandoc
import warnings
warnings.filterwarnings('ignore')
```

sqlite3 : This module provides a simple way to work with SQLite databases, which are a lightweight and serverless type of database often used for local storage in applications.

With the sqlite3 module, we can create, manage, and interact with SQLite databases in our Python code.

# Database Connection

In [4]:
```
connection = sqlite3.connect('E:/STUDY/Data Analyst/Projects/YouTube/Airl
cursor = connection.cursor()
```

sqlite3.connect('travel.sqlite'): This line establishes a connection to an SQLite database file named 'travel.sqlite'.

cursor = connection.cursor(): After establishing the connection, an object named 'cursor' is created. This object allow to execute SQL commands on the database which is used to interact with the database, executing SQL queries and commands.

In [5]:
```
cursor.execute("""SELECT name FROM sqlite_master WHERE type='table';""")
print('List of Tables present in the Database')
table_list = [table[0] for table in cursor.fetchall()]
table_list
```

```
List of Tables present in the Database
```

Out[5]:
```
['aircrafts_data',
 'airports_data',
 'boarding_passes',
 'bookings',
 'flights',
 'seats',
 'ticket_flights',
 'tickets']
```

The above code is effectively retrieving and displaying the list of table names present in the 'travel.sqlite' database.

cursor.fetchall(): This method retrieves all the rows returned by the executed query. In this case, each row contains a single column which is the name of a table.

# Data Exploration

In [6]:
```
Aircraft_Data = pd.read_sql_query(f"""SELECT * FROM aircrafts_data""", co
Aircraft_Data.head()
```

Out[6]:

| | aircraft_code | model | range |
|---|---|---|---|
| **0** | 773 | {"en": "Boeing 777-300", "ru": "Боинг 777-300"} | 11100 |
| **1** | 763 | {"en": "Boeing 767-300", "ru": "Боинг 767-300"} | 7900 |
| **2** | SU9 | {"en": "Sukhoi Superjet-100", "ru": "Сухой Суп... | 3000 |
| **3** | 320 | {"en": "Airbus A320-200", "ru": "Аэробус A320-... | 5700 |
| **4** | 321 | {"en": "Airbus A321-200", "ru": "Аэробус A321-... | 5600 |

The above code uses the 'pd.read_sql_query()' function from the Pandas library to execute a SQL query on the database using the established connection.

In [7]:
```
Airport_Data = pd.read_sql_query(f"""SELECT * FROM airports_data""", conn
Airport_Data.head()
```

Out[7]:

| | airport_code | airport_name | city | coordinates | |
|---|---|---|---|---|---|
| 0 | YKS | {"en": "Yakutsk Airport", "ru": "Якутск"} | {"en": "Yakutsk", "ru": "Якутск"} | (129.77099609375,62.0932998657226562) | A |
| 1 | MJZ | {"en": "Mirny Airport", "ru": "Мирный"} | {"en": "Mirnyj", "ru": "Мирный"} | (114.03900146484375,62.534698486328125) | A |
| 2 | KHV | {"en": "Khabarovsk-Novy Airport", "ru": "Хабар... | {"en": "Khabarovsk", "ru": "Хабаровск"} | (135.18800354004,48.5279998779300001) | Asia/ |
| 3 | PKC | {"en": "Yelizovo Airport", "ru": "Елизово"} | {"en": "Petropavlovsk", "ru": "Петропавловск-К... | (158.453994750976562,53.1679000854492188) | Asia/ |
| 4 | UUS | {"en": "Yuzhno-Sakhalinsk Airport", "ru": "Хом... | {"en": "Yuzhno-Sakhalinsk", "ru": "Южно-Сахали... | (142.718002319335938,46.8886985778808594) | As |

In [8]:
```
Boarding_Pass = pd.read_sql_query(f"""SELECT * FROM boarding_passes""", c
Boarding_Pass.head()
```

Out[8]:

| | ticket_no | flight_id | boarding_no | seat_no |
|---|---|---|---|---|
| 0 | 0005435212351 | 30625 | 1 | 2D |
| 1 | 0005435212386 | 30625 | 2 | 3G |
| 2 | 0005435212381 | 30625 | 3 | 4H |
| 3 | 0005432211370 | 30625 | 4 | 5D |
| 4 | 0005435212357 | 30625 | 5 | 11A |

In [9]:
```
Bookings_Data = pd.read_sql_query(f"""SELECT * FROM bookings """, connect
Bookings_Data.head()
```

Out[9]:

| | book_ref | book_date | total_amount |
|---|---|---|---|
| 0 | 00000F | 2017-07-05 03:12:00+03 | 265700 |
| 1 | 000012 | 2017-07-14 09:02:00+03 | 37900 |
| 2 | 000068 | 2017-08-15 14:27:00+03 | 18100 |
| 3 | 000181 | 2017-08-10 13:28:00+03 | 131800 |
| 4 | 0002D8 | 2017-08-07 21:40:00+03 | 23600 |

In [10]:
```
Flight_Data = pd.read_sql_query(f"""SELECT * FROM flights  """, connectio
Flight_Data
```

Out[10]:

| | flight_id | flight_no | scheduled_departure | scheduled_arrival | departure_airport | arrival_airp |
|---|---|---|---|---|---|---|
| **0** | 1185 | PG0134 | 2017-09-10 09:50:00+03 | 2017-09-10 14:55:00+03 | DME | B |
| **1** | 3979 | PG0052 | 2017-08-25 14:50:00+03 | 2017-08-25 17:35:00+03 | VKO | HM |
| **2** | 4739 | PG0561 | 2017-09-05 12:30:00+03 | 2017-09-05 14:15:00+03 | VKO | AI |
| **3** | 5502 | PG0529 | 2017-09-12 09:50:00+03 | 2017-09-12 11:20:00+03 | SVO | U |
| **4** | 6938 | PG0461 | 2017-09-04 12:25:00+03 | 2017-09-04 13:20:00+03 | SVO | U |
| **...** | ... | ... | ... | ... | ... | |
| **33116** | 33117 | PG0063 | 2017-08-02 19:25:00+03 | 2017-08-02 20:10:00+03 | SKX | S\ |
| **33117** | 33118 | PG0063 | 2017-07-28 19:25:00+03 | 2017-07-28 20:10:00+03 | SKX | S\ |
| **33118** | 33119 | PG0063 | 2017-09-08 19:25:00+03 | 2017-09-08 20:10:00+03 | SKX | S\ |
| **33119** | 33120 | PG0063 | 2017-08-01 19:25:00+03 | 2017-08-01 20:10:00+03 | SKX | S\ |
| **33120** | 33121 | PG0063 | 2017-08-26 19:25:00+03 | 2017-08-26 20:10:00+03 | SKX | S\ |

33121 rows × 10 columns

In [11]:
```
Seat_Data = pd.read_sql_query(f"""SELECT * FROM seats  """, connection)
Seat_Data
```

Out[11]:

| | aircraft_code | seat_no | fare_conditions |
|---|---|---|---|
| **0** | 319 | 2A | Business |
| **1** | 319 | 2C | Business |
| **2** | 319 | 2D | Business |
| **3** | 319 | 2F | Business |
| **4** | 319 | 3A | Business |
| **...** | ... | ... | ... |
| **1334** | 773 | 48H | Economy |
| **1335** | 773 | 48K | Economy |
| **1336** | 773 | 49A | Economy |
| **1337** | 773 | 49C | Economy |
| **1338** | 773 | 49D | Economy |

1339 rows × 3 columns

In [12]:
```
Ticket_Flights = pd.read_sql_query(f"""SELECT * FROM ticket_flights  """,
Ticket_Flights
```

Out[12]:

| | ticket_no | flight_id | fare_conditions | amount |
|---|---|---|---|---|
| **0** | 0005432159776 | 30625 | Business | 42100 |
| **1** | 0005435212351 | 30625 | Business | 42100 |
| **2** | 0005435212386 | 30625 | Business | 42100 |
| **3** | 0005435212381 | 30625 | Business | 42100 |
| **4** | 0005432211370 | 30625 | Business | 42100 |
| **...** | ... | ... | ... | ... |
| **1045721** | 0005435097522 | 32094 | Economy | 5200 |
| **1045722** | 0005435097521 | 32094 | Economy | 5200 |
| **1045723** | 0005435104384 | 32094 | Economy | 5200 |
| **1045724** | 0005435104352 | 32094 | Economy | 5200 |
| **1045725** | 0005435104389 | 32094 | Economy | 5200 |

1045726 rows × 4 columns

In [13]:
```
Ticket_Data = pd.read_sql_query(f" SELECT * FROM tickets ", connection)
Ticket_Data
```

Out[13]:

| | ticket_no | book_ref | passenger_id |
|---|---|---|---|
| **0** | 0005432000987 | 06B046 | 8149 604011 |
| **1** | 0005432000988 | 06B046 | 8499 420203 |
| **2** | 0005432000989 | E170C3 | 1011 752484 |
| **3** | 0005432000990 | E170C3 | 4849 400049 |
| **4** | 0005432000991 | F313DD | 6615 976589 |
| **...** | ... | ... | ... |
| **366728** | 0005435999869 | D730BA | 0474 690760 |
| **366729** | 0005435999870 | D730BA | 6535 751108 |
| **366730** | 0005435999871 | A1AD46 | 1596 156448 |
| **366731** | 0005435999872 | 7B6A53 | 9374 822707 |
| **366732** | 0005435999873 | 7B6A53 | 7380 075822 |

366733 rows × 3 columns

In [14]:

```
for table in table_list:
    print("\ntable: " + table)
    columns_info = connection.execute("PRAGMA table_info({})".format(tabl
    for column in columns_info.fetchall():
        print(column[1:3])
```

```
table: aircrafts_data
('aircraft_code', 'character(3)')
('model', 'jsonb')
('range', 'integer')

table: airports_data
('airport_code', 'character(3)')
('airport_name', 'jsonb')
('city', 'jsonb')
('coordinates', 'point')
('timezone', 'text')

table: boarding_passes
('ticket_no', 'character(13)')
('flight_id', 'integer')
('boarding_no', 'integer')
('seat_no', 'character varying(4)')

table: bookings
('book_ref', 'character(6)')
('book_date', 'timestamp with time zone')
('total_amount', 'numeric(10,2)')

table: flights
('flight_id', 'integer')
('flight_no', 'character(6)')
('scheduled_departure', 'timestamp with time zone')
('scheduled_arrival', 'timestamp with time zone')
('departure_airport', 'character(3)')
('arrival_airport', 'character(3)')
('status', 'character varying(20)')
('aircraft_code', 'character(3)')
('actual_departure', 'timestamp with time zone')
('actual_arrival', 'timestamp with time zone')

table: seats
('aircraft_code', 'character(3)')
('seat_no', 'character varying(4)')
('fare_conditions', 'character varying(10)')

table: ticket_flights
('ticket_no', 'character(13)')
('flight_id', 'integer')
('fare_conditions', 'character varying(10)')
('amount', 'numeric(10,2)')

table: tickets
('ticket_no', 'character(13)')
('book_ref', 'character(6)')
('passenger_id', 'character varying(20)')
```

The above code is effectively printing out the column names and data types for each column in each table listed in the table_list. This provides with the information about the structure of the tables and their columns in the database.

# Data Cleaning

In [15]:
```
for table in table_list:                                    # checking fo
    print(f'\nMissing Values in table : {table}')
    df_table = pd.read_sql_query(f"SELECT * FROM {table}", connection)
    print(df_table.isnull().sum())
```

```
Missing Values in table : aircrafts_data
aircraft_code    0
model            0
range            0
dtype: int64

Missing Values in table : airports_data
airport_code    0
airport_name    0
city            0
coordinates     0
timezone        0
dtype: int64

Missing Values in table : boarding_passes
ticket_no      0
flight_id      0
boarding_no    0
seat_no        0
dtype: int64

Missing Values in table : bookings
book_ref        0
book_date       0
total_amount    0
dtype: int64

Missing Values in table : flights
flight_id              0
flight_no              0
scheduled_departure    0
scheduled_arrival      0
departure_airport      0
arrival_airport        0
status                 0
aircraft_code          0
actual_departure       0
actual_arrival         0
dtype: int64

Missing Values in table : seats
aircraft_code      0
seat_no            0
fare_conditions    0
dtype: int64

Missing Values in table : ticket_flights
ticket_no          0
flight_id          0
fare_conditions    0
amount             0
dtype: int64

Missing Values in table : tickets
ticket_no       0
book_ref        0
```

```
passenger_id    0
dtype: int64
```

The above code provides a summary of missing values for each column in every table present in the database. This can be useful for data quality assessment and for understanding which columns might have incomplete or missing data.

df_table = pd.read_sql_query(f"SELECT * FROM {table}", connection) :

This line reads the data from the current table into a Pandas DataFrame named 'df_table'. The SQL query [f"SELECT * FROM {table}"] retrieves all columns and rows from the specified table.

print(df_table.isnull().sum()) : This line calculates and prints the count of missing values for each column in the 'df_table' DataFrame using the [isnull().sum()] method.

The following method creates a Boolean DataFrame where each cell indicates whether it's a missing value (True) or not (False). The sum() method then sums up the True values (missing values) for each column.

# Basic Analysis

## Q1 - How many planes have more than 100 seats ??

In [16]:
```
pd.read_sql_query(f"""SELECT aircraft_code, COUNT(*) as num_seats FROM sea
                      GROUP BY aircraft_code
                      HAVING num_seats > 100
                      ORDER BY num_seats DESC""", connection)
```

Out[16]:

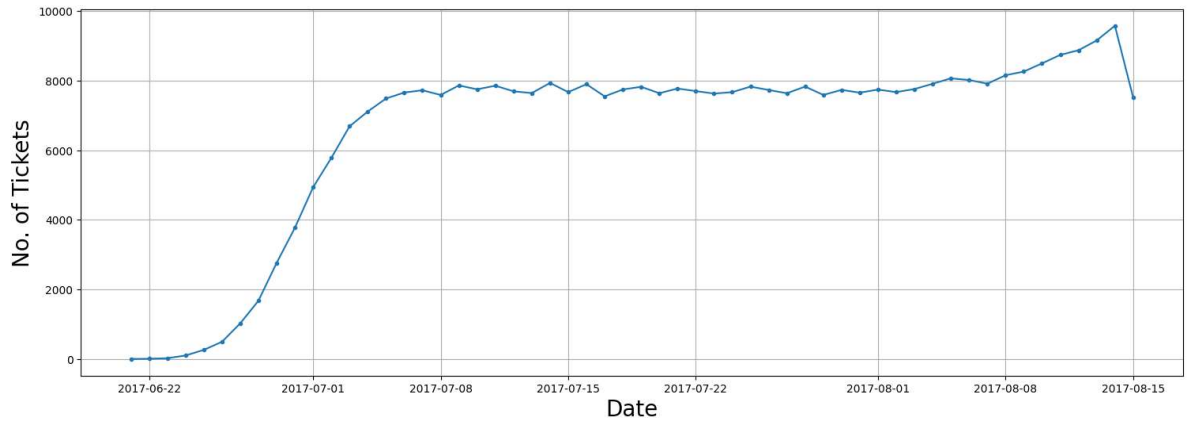
| | aircraft_code | num_seats |
|---|---|---|
| **0** | 773 | 402 |
| **1** | 763 | 222 |
| **2** | 321 | 170 |
| **3** | 320 | 140 |
| **4** | 733 | 130 |
| **5** | 319 | 116 |

By using the above query with 'pd.read_sql_query()', the desired data is retrieved from the database and a Pandas DataFrame is created containing the aircraft codes and the count of seats for each aircraft that has more than 100 seats. The DataFrame will be returned as the output of this line of code.

This query helps in identifying aircraft models that have a significant number of seats, which might be useful for analyzing the capacity of different aircrafts.

## Q2 - How the number of booked tickets and total amount earned changes with the time ??

In [17]:
```
Tickets = pd.read_sql_query(f"""SELECT *
                                FROM tickets
                                INNER JOIN bookings
                                ON tickets.book_ref=bookings.book_ref;""",

Tickets['book_date'] = pd.to_datetime(Tickets['book_date'])
Tickets['date'] = Tickets['book_date'].dt.date

x =  Tickets.groupby('date')[['date']].count()

plt.figure(figsize = (18,6))               # creating a new figure with a
plt.plot(x.index,x['date'], marker = '.')

plt.xlabel('Date', fontsize = 20)
plt.ylabel('No. of Tickets', fontsize = 20)

plt.grid('b')       # The 'b' argument specifies that the grid lines shou
plt.show()
```



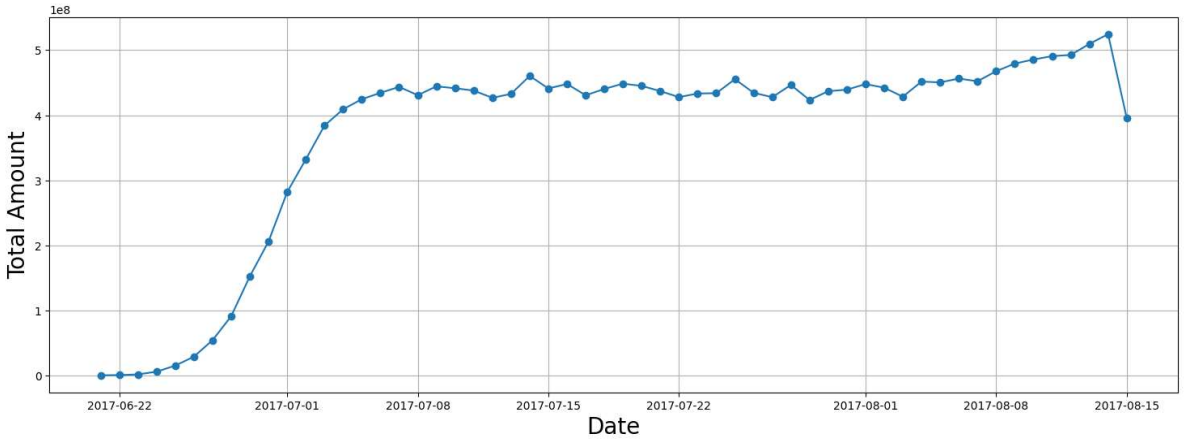

The 1st part of the code retrieves data from the "tickets" and "bookings" tables by performing an inner join on the "book_ref" column. It fetches all columns from both tables for the joined rows and stores the result in the 'Tickets' DataFrame.

The 2nd part of the code convert the 'book_date' column to a Pandas datetime format and then extracts the date component from it. The extracted date is stored in a new column named 'date'.

The 3rd part of the code performs a grouping operation on the 'date' column to count the number of occurrences (tickets) for each date.

In [18]:

```
bookings = pd.read_sql_query(f"""SELECT * FROM bookings""", connection)

bookings['book_date'] = pd.to_datetime(bookings['book_date'])
bookings['date'] = bookings['book_date'].dt.date

y =  bookings.groupby('date')[['total_amount']].sum()

plt.figure(figsize = (18,6))
plt.plot(y.index,y['total_amount'], marker = 'o')

plt.xlabel('Date', fontsize = 20)
plt.ylabel('Total Amount', fontsize = 20)

plt.grid('b')
plt.show()
```



## Q3 - Calculate the average charges for each aircraft with different fare conditions.

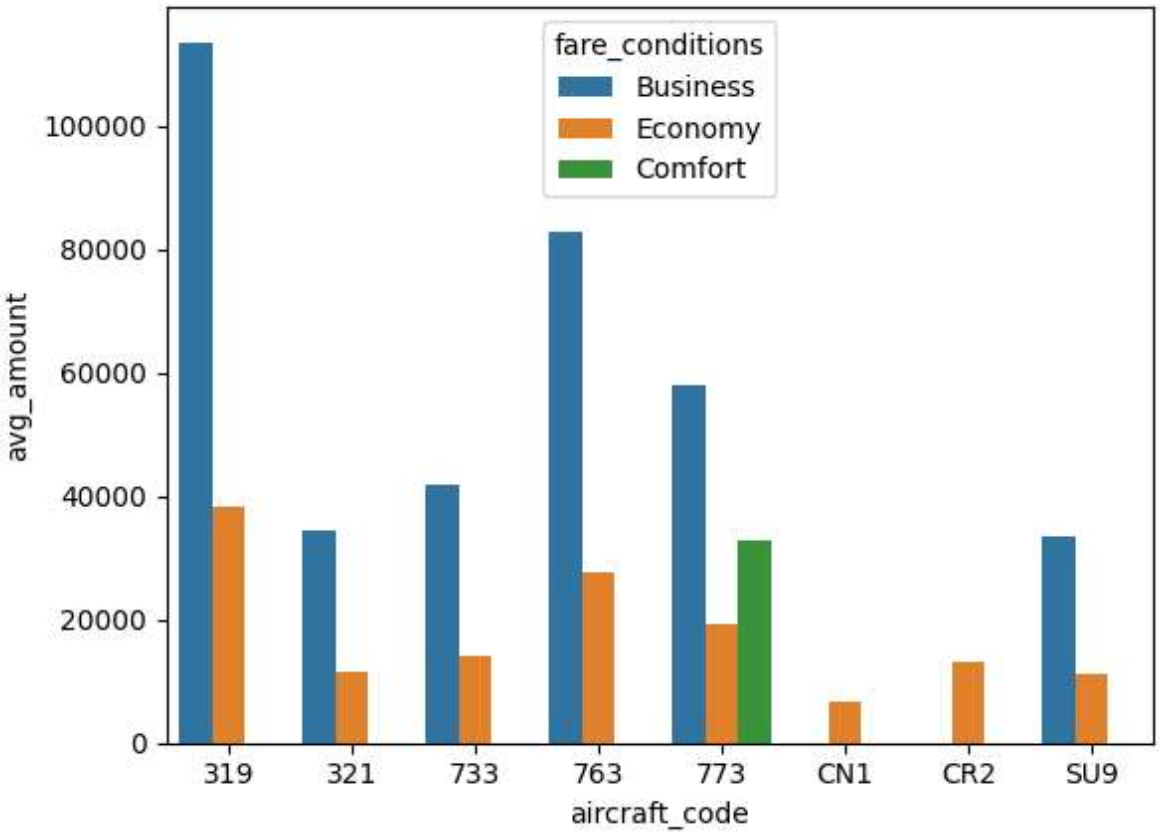
In [19]:
```
df = pd.read_sql_query(f"""SELECT fare_conditions, aircraft_code, AVG(amou
                           FROM ticket_flights
                           JOIN flights
                           ON ticket_flights.flight_id=flights.flight_id
                           GROUP BY aircraft_code, fare_conditions""", co
df
```

Out[19]:

| | fare_conditions | aircraft_code | avg_amount |
|---|---|---|---|
| **0** | Business | 319 | 113550.557703 |
| **1** | Economy | 319 | 38311.402347 |
| **2** | Business | 321 | 34435.662664 |
| **3** | Economy | 321 | 11534.974764 |
| **4** | Business | 733 | 41865.626175 |
| **5** | Economy | 733 | 13985.152000 |
| **6** | Business | 763 | 82839.842866 |
| **7** | Economy | 763 | 27594.721829 |
| **8** | Business | 773 | 57779.909435 |
| **9** | Comfort | 773 | 32740.552889 |
| **10** | Economy | 773 | 19265.225693 |
| **11** | Economy | CN1 | 6568.552345 |
| **12** | Economy | CR2 | 13207.661102 |
| **13** | Business | SU9 | 33487.849829 |
| **14** | Economy | SU9 | 11220.183400 |

In [20]:
```
1 sns.barplot(data = df, x = 'aircraft_code', y ='avg_amount', hue = 'fare_
```

Out[20]: <Axes: xlabel='aircraft_code', ylabel='avg_amount'>




Here, the Seaborn's barplot() function is used to create the bar plot.

The resulting bar plot will display bars for each aircraft code, and within each bar, there will be different colored segments representing the average fare amounts for different fare conditions. The hue parameter ensures that each fare condition gets a distinct color.

# Analyzing the occupancy rate

## 1 - For each aircraft, calculate the total revenue per year and the average revenue per ticket.

In [21]:
```
pd.read_sql_query(f"""SELECT aircraft_code, total_revenue, ticket_count, t
                      FROM
                        ( SELECT aircraft_code, COUNT(*) as ticket_count,
                          FROM ticket_flights
                          JOIN flights
                          ON ticket_flights.flight_id=flights.flight_id
                          GROUP BY aircraft_code )""", connection)
```

Out[21]:

| | aircraft_code | total_revenue | ticket_count | avg_revenue_per_ticket |
|---|---|---|---|---|
| **0** | 319 | 2706163100 | 52853 | 51201 |
| **1** | 321 | 1638164100 | 107129 | 15291 |
| **2** | 733 | 1426552100 | 86102 | 16568 |
| **3** | 763 | 4371277100 | 124774 | 35033 |
| **4** | 773 | 3431205500 | 144376 | 23765 |
| **5** | CN1 | 96373800 | 14672 | 6568 |
| **6** | CR2 | 1982760500 | 150122 | 13207 |
| **7** | SU9 | 5114484700 | 365698 | 13985 |

## 2 - Calculate the average occupancy per aircraft.

In [22]:
```
occupancy_rate = pd.read_sql_query(f"""SELECT a.aircraft_code, AVG(a.seats
                                       AVG(a.seats_count)/b.num_sea
                                  FROM (
                                       SELECT aircraft_code, flight
                                       FROM boarding_passes
                                       INNER JOIN flights
                                       ON boarding_passes.flight_id
                                       GROUP BY aircraft_code, flig
                                  ) as a

                                  INNER JOIN

                                  (
                                       SELECT aircraft_code, COUNT(
                                       FROM seats
                                       GROUP BY aircraft_code
                                  ) as b

                                  ON a.aircraft_code = b.aircraft_cod
                                  GROUP BY a.aircraft_code""", connec
occupancy_rate
```

Out[22]:

| | aircraft_code | booked_seats | num_seats | occupancy_rate |
|---|---|---|---|---|
| **0** | 319 | 53.583181 | 116 | 0.461924 |
| **1** | 321 | 88.809231 | 170 | 0.522407 |
| **2** | 733 | 80.255462 | 130 | 0.617350 |
| **3** | 763 | 113.937294 | 222 | 0.513231 |
| **4** | 773 | 264.925806 | 402 | 0.659019 |
| **5** | CN1 | 6.004431 | 12 | 0.500369 |
| **6** | CR2 | 21.482847 | 50 | 0.429657 |
| **7** | SU9 | 56.812113 | 97 | 0.585692 |

## 3 - Calculate by how much the total annual turnover could increase by giving all aircraft a 10% higher occupancy rate.

In [23]:
```
occupancy_rate['Increased occupancy rate'] = occupancy_rate['occupancy_ra
occupancy_rate
```

Out[23]:

| | aircraft_code | booked_seats | num_seats | occupancy_rate | Increased occupancy rate |
|---|---|---|---|---|---|
| **0** | 319 | 53.583181 | 116 | 0.461924 | 0.508116 |
| **1** | 321 | 88.809231 | 170 | 0.522407 | 0.574648 |
| **2** | 733 | 80.255462 | 130 | 0.617350 | 0.679085 |
| **3** | 763 | 113.937294 | 222 | 0.513231 | 0.564554 |
| **4** | 773 | 264.925806 | 402 | 0.659019 | 0.724921 |
| **5** | CN1 | 6.004431 | 12 | 0.500369 | 0.550406 |
| **6** | CR2 | 21.482847 | 50 | 0.429657 | 0.472623 |
| **7** | SU9 | 56.812113 | 97 | 0.585692 | 0.644261 |

In [24]:
```
total_revenue = pd.read_sql_query("""SELECT aircraft_code, SUM(amount) as
                                     FROM ticket_flights
                                     JOIN flights
                                     ON ticket_flights.flight_id=flights.
                                     GROUP BY aircraft_code""", connectio

total_revenue
```

Out[24]:

| | aircraft_code | total_revenue |
|---|---|---|
| **0** | 319 | 2706163100 |
| **1** | 321 | 1638164100 |
| **2** | 733 | 1426552100 |
| **3** | 763 | 4371277100 |
| **4** | 773 | 3431205500 |
| **5** | CN1 | 96373800 |
| **6** | CR2 | 1982760500 |
| **7** | SU9 | 5114484700 |

In [25]:
```
occupancy_rate['Increased Total Annual Turnover'] = (total_revenue['total_

occupancy_rate
```

Out[25]:

| | aircraft_code | booked_seats | num_seats | occupancy_rate | Increased occupancy rate | Increased Total Annual Turnover |
|---|---|---|---|---|---|---|
| **0** | 319 | 53.583181 | 116 | 0.461924 | 0.508116 | 2.976779e+09 |
| **1** | 321 | 88.809231 | 170 | 0.522407 | 0.574648 | 1.801981e+09 |
| **2** | 733 | 80.255462 | 130 | 0.617350 | 0.679085 | 1.569207e+09 |
| **3** | 763 | 113.937294 | 222 | 0.513231 | 0.564554 | 4.808405e+09 |
| **4** | 773 | 264.925806 | 402 | 0.659019 | 0.724921 | 3.774326e+09 |
| **5** | CN1 | 6.004431 | 12 | 0.500369 | 0.550406 | 1.060112e+08 |
| **6** | CR2 | 21.482847 | 50 | 0.429657 | 0.472623 | 2.181037e+09 |
| **7** | SU9 | 56.812113 | 97 | 0.585692 | 0.644261 | 5.625933e+09 |

In [26]:
```
pd.set_option('display.float_format', str)
```

The above command sets a display option in Pandas to format floating-point numbers as strings when they are displayed in DataFrames.

By using this command, any floating-point numbers in the DataFrame will be displayed as strings in their original format, without any decimal formatting. This can be useful when we want to prevent Pandas from automatically formatting floating-point numbers with a certain number of decimal places as seen in above column 'Increased Total Annual Turnover'

Keep in mind that this option will affect the display of floating-point numbers across all DataFrames in the session, so make sure to set it back to the default if you want to revert to the standard formatting.

In [27]:
```
occupancy_rate['Increased Total Annual Turnover'] = (total_revenue['total_

occupancy_rate
```

Out[27]:

| | aircraft_code | booked_seats | num_seats | occupancy_rate | Increased occupancy rate | |
|---|---|---|---|---|---|---|
| **0** | 319 | 53.58318098720292 | 116 | 0.46192397402761143 | 0.5081163714303726 | |
| **1** | 321 | 88.80923076923077 | 170 | 0.5224072398190045 | 0.574647963800905 | |
| **2** | 733 | 80.25546218487395 | 130 | 0.617349709114415 | 0.6790846800258565 | 156 |
| **3** | 763 | 113.93729372937294 | 222 | 0.5132310528350132 | 0.5645541581185146 | |
| **4** | 773 | 264.9258064516129 | 402 | 0.659019419033863 | 0.7249213609372492 | |
| **5** | CN1 | 6.004431314623338 | 12 | 0.5003692762186115 | 0.5504062038404727 | 106 |
| **6** | CR2 | 21.48284690220174 | 50 | 0.42965693804403476 | 0.4726226318484382 | |
| **7** | SU9 | 56.81211267605634 | 97 | 0.5856918832583128 | 0.644261071584144 | 56 |