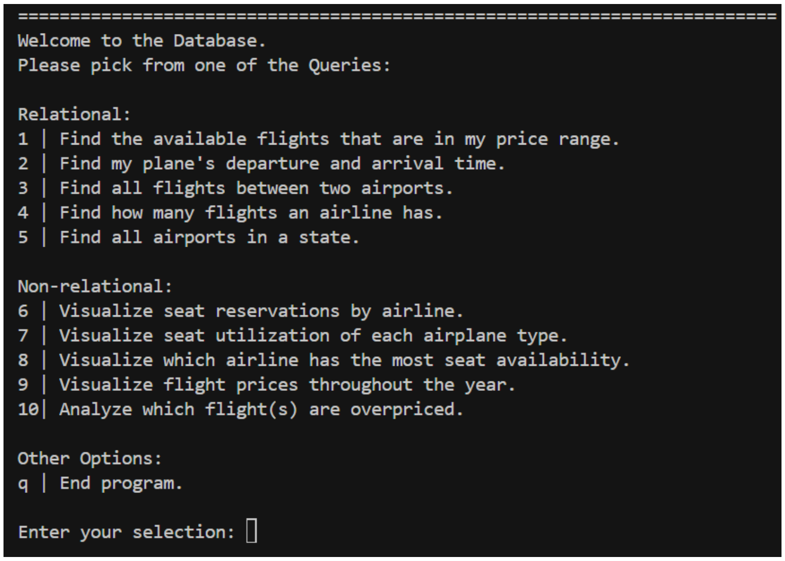
Connor Hoang (28239138)
EECS 118
10 December 2024

Term Project

Intro

Using the provided flight dataset, this project will delve into the relational and non-relational queries made for the set. The relational queries will all use SQL to query the data, while non-relational queries will use pi chart, bar plot, and scatter plot visualizations, and outlier predictions to query the data.

Starting menu



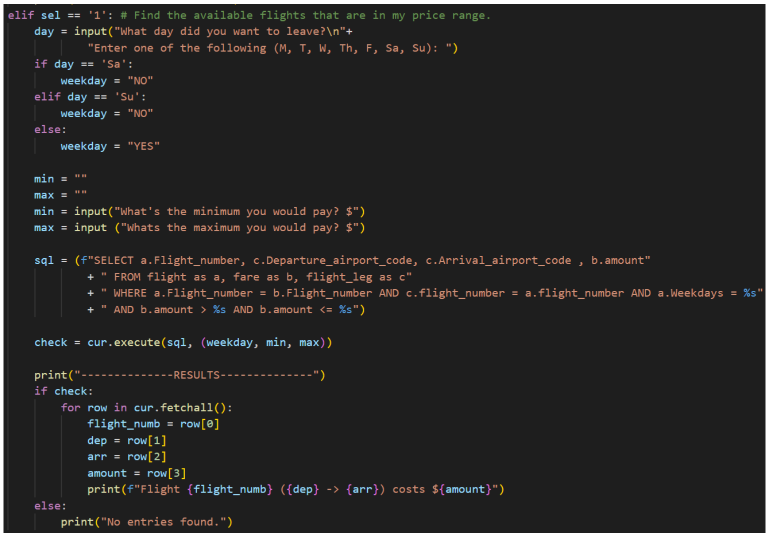
```
=========================================================================
Welcome to the Database.
Please pick from one of the Queries:

Relational:
1 | Find the available flights that are in my price range.
2 | Find my plane's departure and arrival time.
3 | Find all flights between two airports.
4 | Find how many flights an airline has.
5 | Find all airports in a state.

Non-relational:
6 | Visualize seat reservations by airline.
7 | Visualize seat utilization of each airplane type.
8 | Visualize which airline has the most seat availability.
9 | Visualize flight prices throughout the year.
10| Analyze which flight(s) are overpriced.

Other Options:
q | End program.

Enter your selection: 
```

Upon starting the program, the user is greeted with the options shown above and organized to display the relational and non-relational queries. The user is also given the option ‘q’ to end the program smoothly.

Relational Queries

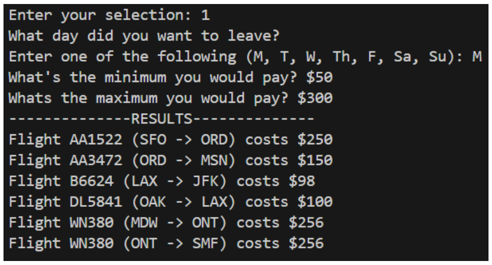
<u>Find the available flights that are in my price range.</u>

Code

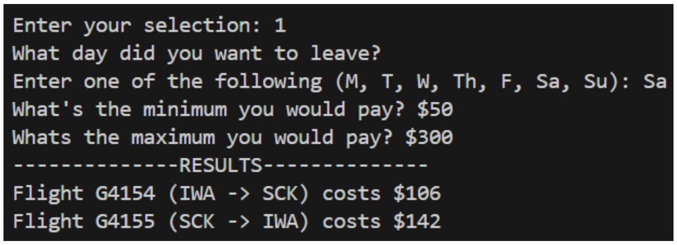


```python
elif sel == '1': # Find the available flights that are in my price range.
    day = input("What day did you want to leave?\n"+
            "Enter one of the following (M, T, W, Th, F, Sa, Su): ")
    if day == 'Sa':
        weekday = "NO"
    elif day == 'Su':
        weekday = "NO"
    else:
        weekday = "YES"

    min = ""
    max = ""
    min = input("What's the minimum you would pay? $")
    max = input ("Whats the maximum you would pay? $")

    sql = (f"SELECT a.Flight_number, c.Departure_airport_code, c.Arrival_airport_code , b.amount"
            + " FROM flight as a, fare as b, flight_leg as c"
            + " WHERE a.Flight_number = b.Flight_number AND c.flight_number = a.flight_number AND a.Weekdays = %s"
            + " AND b.amount > %s AND b.amount <= %s")

    check = cur.execute(sql, (weekday, min, max))

    print("-------------RESULTS-------------")
    if check:
        for row in cur.fetchall():
            flight_numb = row[0]
            dep = row[1]
            arr = row[2]
            amount = row[3]
            print(f"Flight {flight_numb} ({dep} -> {arr}) costs ${amount}")
    else:
        print("No entries found.")
```

Output



```
Enter your selection: 1
What day did you want to leave?
Enter one of the following (M, T, W, Th, F, Sa, Su): M
What's the minimum you would pay? $50
Whats the maximum you would pay? $300
-------------RESULTS-------------
Flight AA1522 (SFO -> ORD) costs $250
Flight AA3472 (ORD -> MSN) costs $150
Flight B6624 (LAX -> JFK) costs $98
Flight DL5841 (OAK -> LAX) costs $100
Flight WN380 (MDW -> ONT) costs $256
Flight WN380 (ONT -> SMF) costs $256
```

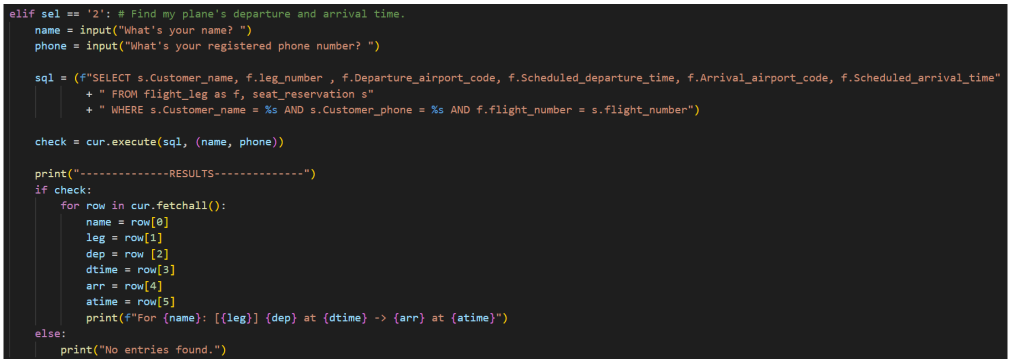


```
Enter your selection: 1
What day did you want to leave?
Enter one of the following (M, T, W, Th, F, Sa, Su): Sa
What's the minimum you would pay? $50
Whats the maximum you would pay? $300
-------------RESULTS-------------
Flight G4154 (IWA -> SCK) costs $106
Flight G4155 (SCK -> IWA) costs $142
```

After giving the program the day the user would like to leave and their price range, the query will find all available flights offered and display their flight number, airport of departure, airport of arrival, and how much it would cost to book that flight. The image above shows two different days used with the same price range. We can see that there are more flights available during the weekdays with this price range compared to the weekends.

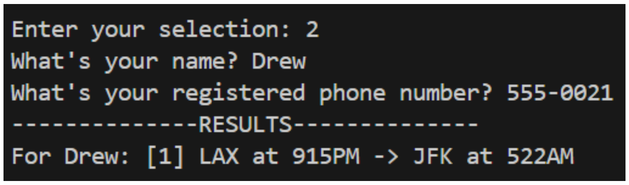
Find my plane's departure and arrival time.

Code

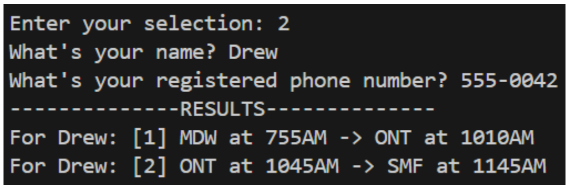


```
elif sel == '2': # Find my plane's departure and arrival time.
    name = input("What's your name? ")
    phone = input("What's your registered phone number? ")

    sql = (f"SELECT s.Customer_name, f.leg_number , f.Departure_airport_code, f.Scheduled_departure_time, f.Arrival_airport_code, f.Scheduled_arrival_time"
            + " FROM flight_leg as f, seat_reservation s"
            + " WHERE s.Customer_name = %s AND s.Customer_phone = %s AND f.flight_number = s.flight_number")

    check = cur.execute(sql, (name, phone))

    print("-------------RESULTS-------------")
    if check:
        for row in cur.fetchall():
            name = row[0]
            leg = row[1]
            dep = row [2]
            dtime = row[3]
            arr = row[4]
            atime = row[5]
            print(f"For {name}: [{leg}] {dep} at {dtime} -> {arr} at {atime}")
    else:
        print("No entries found.")
```

Output



```
Enter your selection: 2
What's your name? Drew
What's your registered phone number? 555-0021
--------------RESULTS--------------
For Drew: [1] LAX at 915PM -> JFK at 522AM
```

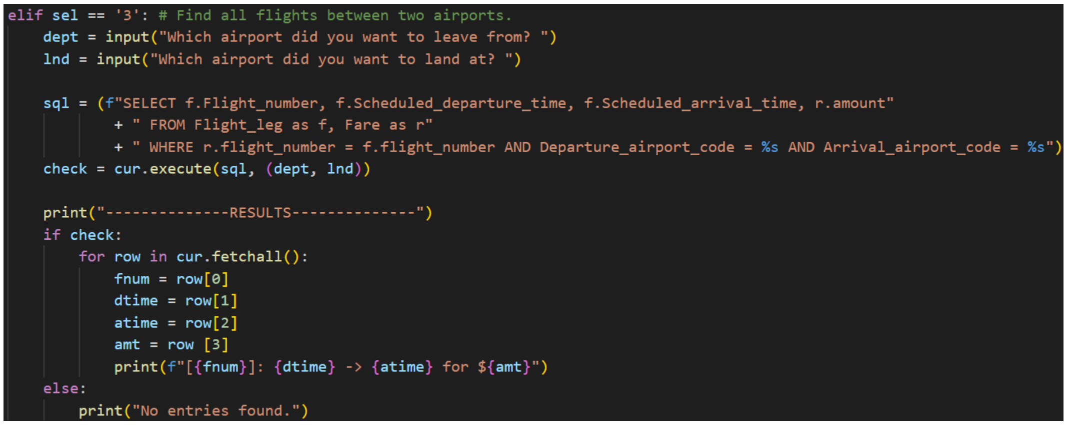


```
Enter your selection: 2
What's your name? Drew
What's your registered phone number? 555-0042
--------------RESULTS--------------
For Drew: [1] MDW at 755AM -> ONT at 1010AM
For Drew: [2] ONT at 1045AM -> SMF at 1145AM
```

After giving the program the user's name and phone number, the query will find all flights under their name. Their phone number is used as a specifier to ensure the correct person is identified during the query. After the query runs, the program will output the person's flight by flight leg, the departure airport and time, and the arrival airport and time of each flight leg. The screenshots above shows the output after it's run through two different people (phone numbers) with the same name (Drew).

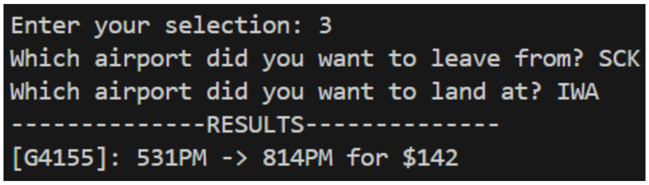
Find all flights between two airports.

Code



```
elif sel == '3': # Find all flights between two airports.
    dept = input("Which airport did you want to leave from? ")
    lnd = input("Which airport did you want to land at? ")

    sql = (f"SELECT f.Flight_number, f.Scheduled_departure_time, f.Scheduled_arrival_time, r.amount"
            + " FROM Flight_leg as f, Fare as r"
            + " WHERE r.flight_number = f.flight_number AND Departure_airport_code = %s AND Arrival_airport_code = %s")
    check = cur.execute(sql, (dept, lnd))

    print("-------------RESULTS-------------")
    if check:
        for row in cur.fetchall():
            fnum = row[0]
            dtime = row[1]
            atime = row[2]
            amt = row [3]
            print(f"[{fnum}]: {dtime} -> {atime} for ${amt}")
    else:
        print("No entries found.")
```

Output

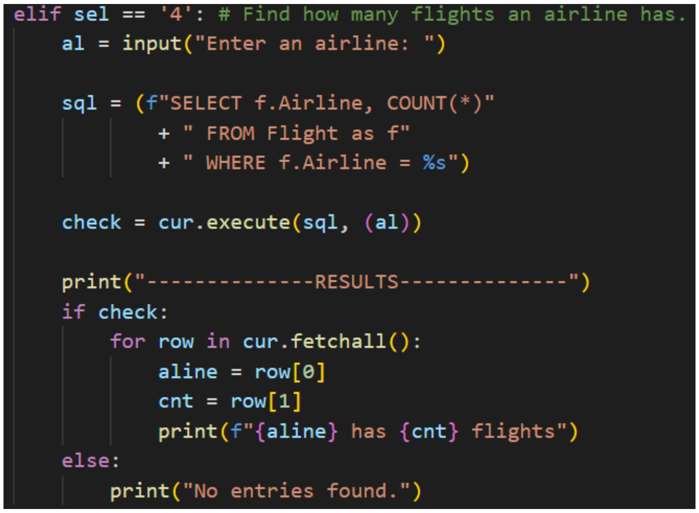


```
Enter your selection: 3
Which airport did you want to leave from? SCK
Which airport did you want to land at? IWA
--------------RESULTS--------------
[G4155]: 531PM -> 814PM for $142
```

After giving the program which airports the user would like to leave from and land at, the query will find all flights connected to both airports. The output will show each flight number, departure and arrival time, and the price of that flight. Given the dataset, every input would only show one output, just different flight numbers, times and prices. If the dataset were larger, there would be more to display for each airport.

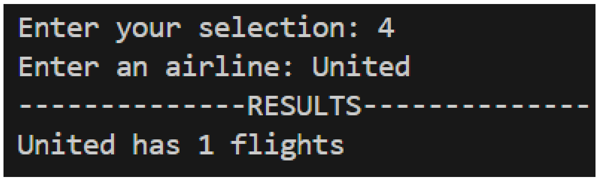
Find how many flights an airline has.

Code

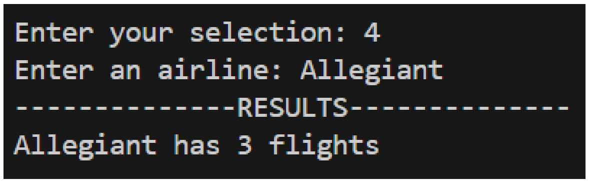


```
elif sel == '4': # Find how many flights an airline has.
    al = input("Enter an airline: ")

    sql = (f"SELECT f.Airline, COUNT(*)"
            + " FROM Flight as f"
            + " WHERE f.Airline = %s")

    check = cur.execute(sql, (al))

    print("--------------RESULTS--------------")
    if check:
        for row in cur.fetchall():
            aline = row[0]
            cnt = row[1]
            print(f"{aline} has {cnt} flights")
    else:
        print("No entries found.")
```

Output



```
Enter your selection: 4
Enter an airline: United
--------------RESULTS--------------
United has 1 flights
```

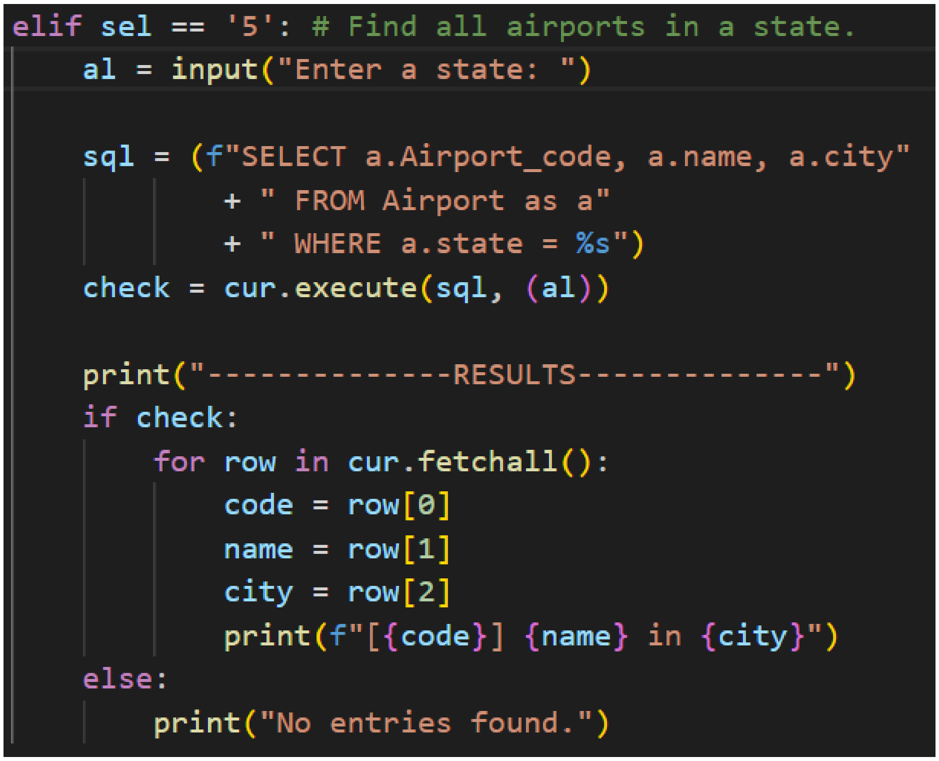


```
Enter your selection: 4
Enter an airline: Allegiant
--------------RESULTS--------------
Allegiant has 3 flights
```

After a user inputs an airline’s name, the query will output the number of flights that belong to that airline. Given the data set, there are only a limited number of flights that are tracked and therefore the results won’t show any large data points. This query would be important for those tracking the amount of flights to test against an airline’s mileage or fuel usage.

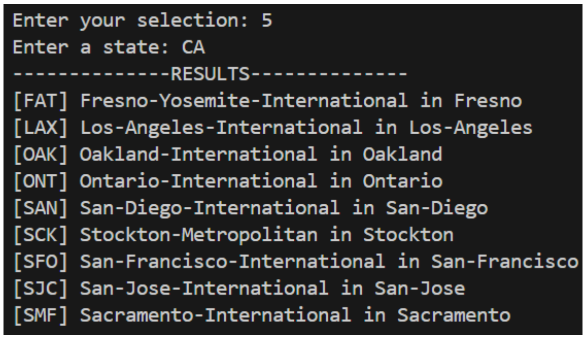
Find all airports in a state.

Code

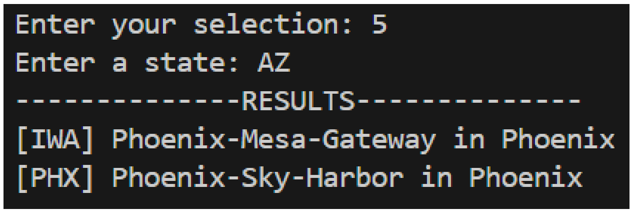


```
elif sel == '5': # Find all airports in a state.
    al = input("Enter a state: ")

    sql = (f"SELECT a.Airport_code, a.name, a.city"
            + " FROM Airport as a"
            + " WHERE a.state = %s")
    check = cur.execute(sql, (al))

    print("--------------RESULTS--------------")
    if check:
        for row in cur.fetchall():
            code = row[0]
            name = row[1]
            city = row[2]
            print(f"[{code}] {name} in {city}")
    else:
        print("No entries found.")
```

Output



```
Enter your selection: 5
Enter a state: CA
-------------RESULTS-------------
[FAT] Fresno-Yosemite-International in Fresno
[LAX] Los-Angeles-International in Los-Angeles
[OAK] Oakland-International in Oakland
[ONT] Ontario-International in Ontario
[SAN] San-Diego-International in San-Diego
[SCK] Stockton-Metropolitan in Stockton
[SFO] San-Francisco-International in San-Francisco
[SJC] San-Jose-International in San-Jose
[SMF] Sacramento-International in Sacramento
```



```
Enter your selection: 5
Enter a state: AZ
--------------RESULTS--------------
[IWA] Phoenix-Mesa-Gateway in Phoenix
[PHX] Phoenix-Sky-Harbor in Phoenix
```

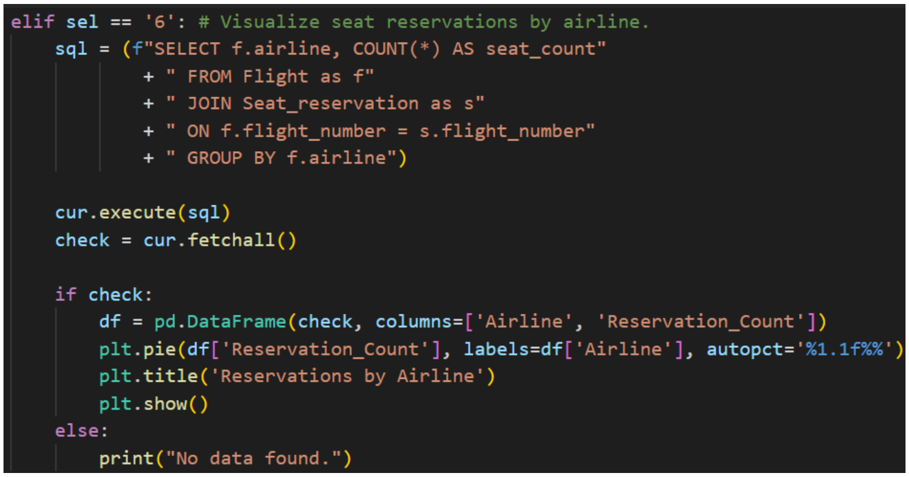
After the user inputs which state they'd like to search, the query will output every airport code, airport name, and city the airports are located in that state. The above screenshots showcase a user searching all airports in California and Arizona.

Non-relational Queries

As a disclaimer: given the dataset of flights, the data used and output for my non-relational queries aren't grand and don't have a lot of volume.

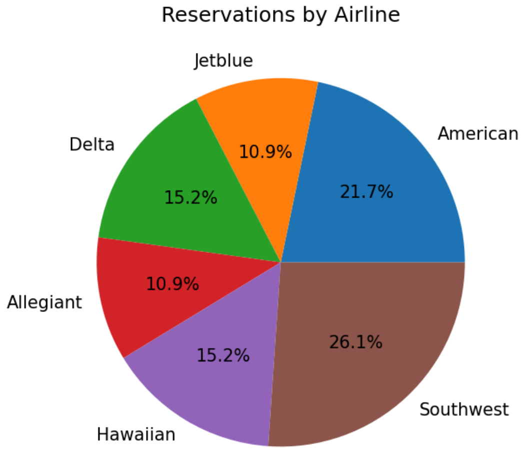
<u>Visualize seat reservations by airline.</u>

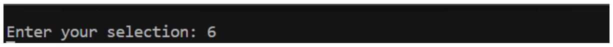
Code



```
elif sel == '6': # Visualize seat reservations by airline.
    sql = (f"SELECT f.airline, COUNT(*) AS seat_count"
            + " FROM Flight as f"
            + " JOIN Seat_reservation as s"
            + " ON f.flight_number = s.flight_number"
            + " GROUP BY f.airline")

    cur.execute(sql)
    check = cur.fetchall()

    if check:
        df = pd.DataFrame(check, columns=['Airline', 'Reservation_Count'])
        plt.pie(df['Reservation_Count'], labels=df['Airline'], autopct='%1.1f%%')
        plt.title('Reservations by Airline')
        plt.show()
    else:
        print("No data found.")
```

Output



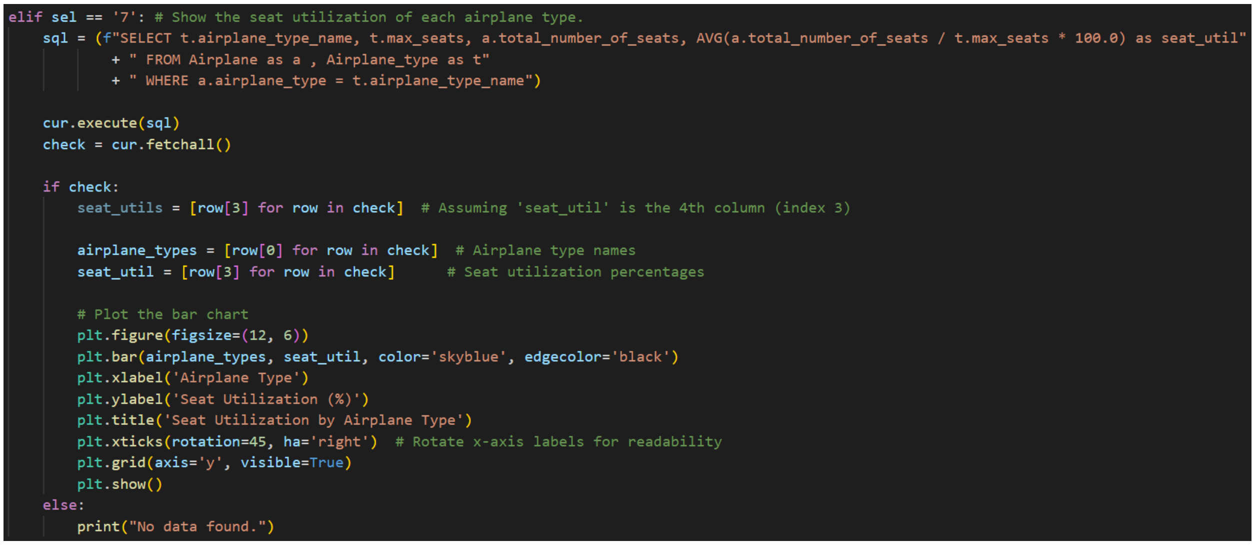


```
Enter your selection: 6
```

After selecting query 6, the program will first perform an SQL query to get the seat reservation count of all airlines. This count from each airline will then be compared against the others and placed into a pi chart which visualizes which airlines are being more prominently used by customers. The pi chart overall shows the frequency of airlines used by consumers.

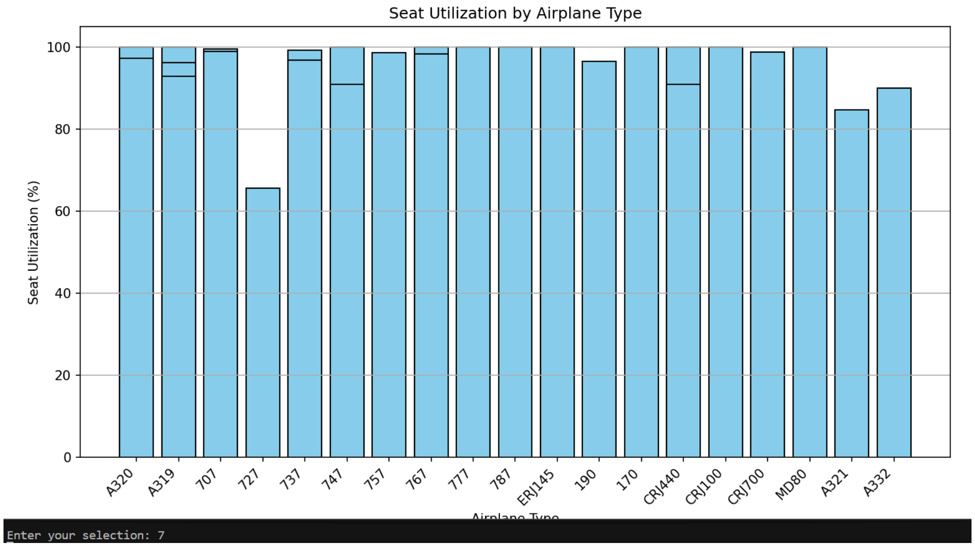
Visualize seat utilization of each airplane type.

Code



```
elif sel == '7': # Show the seat utilization of each airplane type.
    sql = (f"SELECT t.airplane_type_name, t.max_seats, a.total_number_of_seats, AVG(a.total_number_of_seats / t.max_seats * 100.0) as seat_util"
            + " FROM Airplane as a , Airplane_type as t"
            + " WHERE a.airplane_type = t.airplane_type_name")

    cur.execute(sql)
    check = cur.fetchall()

    if check:
        seat_utils = [row[3] for row in check]  # Assuming 'seat_util' is the 4th column (index 3)

        airplane_types = [row[0] for row in check]  # Airplane type names
        seat_util = [row[3] for row in check]      # Seat utilization percentages

        # Plot the bar chart
        plt.figure(figsize=(12, 6))
        plt.bar(airplane_types, seat_util, color='skyblue', edgecolor='black')
        plt.xlabel('Airplane Type')
        plt.ylabel('Seat Utilization (%)')
        plt.title('Seat Utilization by Airplane Type')
        plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
        plt.grid(axis='y', visible=True)
        plt.show()
    else:
        print("No data found.")
```

Output

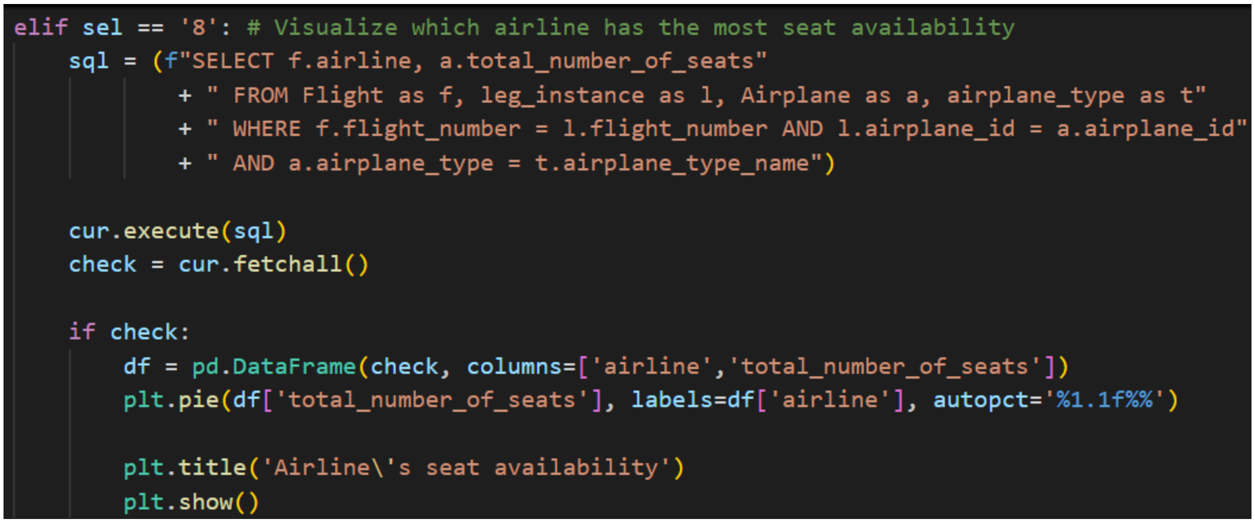


```
Enter your selection: 7
```

After selecting query 7, the program will query the dataset to find the airplane types and compare their max seats to their available seats. This histogram shows the seat utilization of each airplane type. To read this histogram, we look at the plane type (ie A319) and see each line in the bar to be the seat utilization (ie the top/max utilization of A319 is 100% and the bottom/min utilization is ~90%).

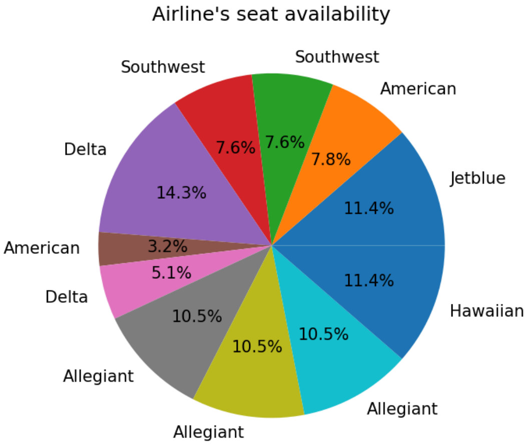
Visualize which airline has the most seat availability.

Code



```
elif sel == '8': # Visualize which airline has the most seat availability
    sql = (f"SELECT f.airline, a.total_number_of_seats"
            + " FROM Flight as f, leg_instance as l, Airplane as a, airplane_type as t"
            + " WHERE f.flight_number = l.flight_number AND l.airplane_id = a.airplane_id"
            + " AND a.airplane_type = t.airplane_type_name")

    cur.execute(sql)
    check = cur.fetchall()

    if check:
        df = pd.DataFrame(check, columns=['airline','total_number_of_seats'])
        plt.pie(df['total_number_of_seats'], labels=df['airline'], autopct='%1.1f%%')

        plt.title('Airline\'s seat availability')
        plt.show()
```

Output

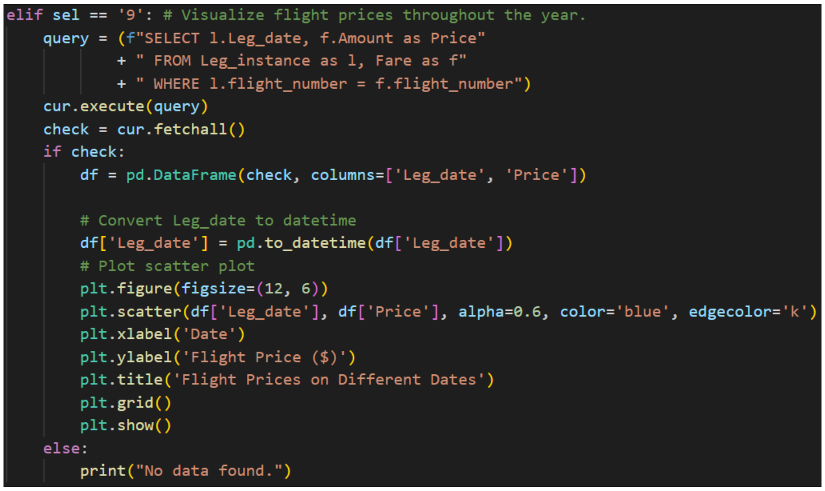


```
Enter your selection: 8
```

After selecting query 8, the program will find the total number of seats each airline provides and displays it in a pi chart. This reveals which airline provides the most amount of seats, whether the mass amount of seats means that the airline’s planes are typically bigger, or if the seat availability is due to a normal-sized plane with more cramped seating for its customers. If the data set had been able to provide the plane’s size vs customer rating, these could be graphed to further predict customer satisfaction.

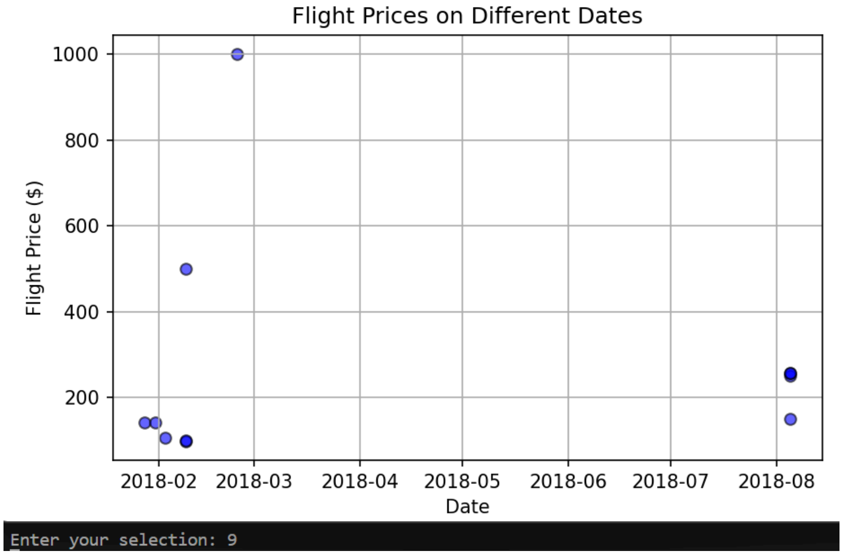
Visualize flight prices throughout the year.

Code



```
elif sel == '9': # Visualize flight prices throughout the year.
    query = (f"SELECT l.Leg_date, f.Amount as Price"
            + " FROM Leg_instance as l, Fare as f"
            + " WHERE l.flight_number = f.flight_number")
    cur.execute(query)
    check = cur.fetchall()
    if check:
        df = pd.DataFrame(check, columns=['Leg_date', 'Price'])

        # Convert Leg_date to datetime
        df['Leg_date'] = pd.to_datetime(df['Leg_date'])
        # Plot scatter plot
        plt.figure(figsize=(12, 6))
        plt.scatter(df['Leg_date'], df['Price'], alpha=0.6, color='blue', edgecolor='k')
        plt.xlabel('Date')
        plt.ylabel('Flight Price ($)')
        plt.title('Flight Prices on Different Dates')
        plt.grid()
        plt.show()
    else:
        print("No data found.")
```

Output

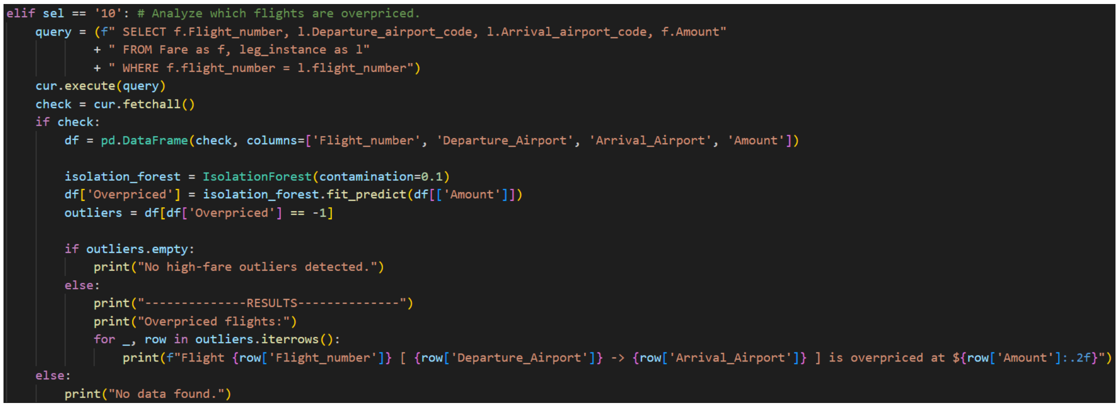


```
Enter your selection: 9
```

After selecting query 9, the program will collect the prices from each flight and the flight's dates. It would then proceed to graph these data points onto a scatter plot to visualize the typical growth and decay of flight prices depending on specific times of the year (ie higher prices during the holidays due to higher demand), which is similar to the skyrocketed price before 2018-03 in the graph. Due to the lack of data points in the set, the graph looks less impressive, but the use for the query would be highly beneficial for larger datasets.

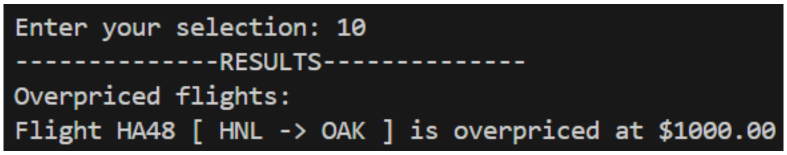
Analyze which flight(s) are overpriced.

Code



```python
elif sel == '10': # Analyze which flights are overpriced.
    query = (f" SELECT f.Flight_number, l.Departure_airport_code, l.Arrival_airport_code, f.Amount"
            + " FROM Fare as f, leg_instance as l"
            + " WHERE f.flight_number = l.flight_number")
    cur.execute(query)
    check = cur.fetchall()
    if check:
        df = pd.DataFrame(check, columns=['Flight_number', 'Departure_Airport', 'Arrival_Airport', 'Amount'])

        isolation_forest = IsolationForest(contamination=0.1)
        df['Overpriced'] = isolation_forest.fit_predict(df[['Amount']])
        outliers = df[df['Overpriced'] == -1]

        if outliers.empty:
            print("No high-fare outliers detected.")
        else:
            print("--------------RESULTS--------------")
            print("Overpriced flights:")
            for _, row in outliers.iterrows():
                print(f"Flight {row['Flight_number']} [ {row['Departure_Airport']} -> {row['Arrival_Airport']} ] is overpriced at ${row['Amount']:.2f}")
    else:
        print("No data found.")
```

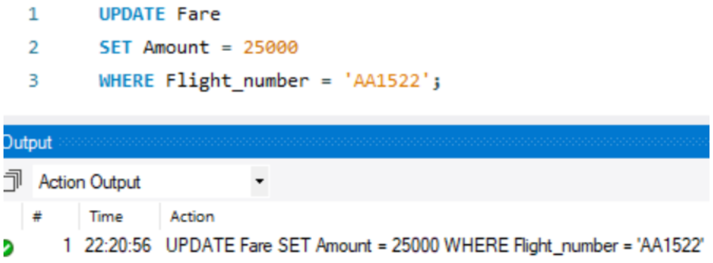
Output



```
Enter your selection: 10
--------------RESULTS--------------
Overpriced flights:
Flight HA48 [ HNL -> OAK ] is overpriced at $1000.00
```

After selecting query 10, the program will collect the data of all flights and their correlating prices to locate an outlying flight price. The output will show the flight number and the flight departure and arrival airports along with the price. This query can show consumers the flights to avoid due to overpricing or they can help companies decide more desirable flight prices.

Data Change

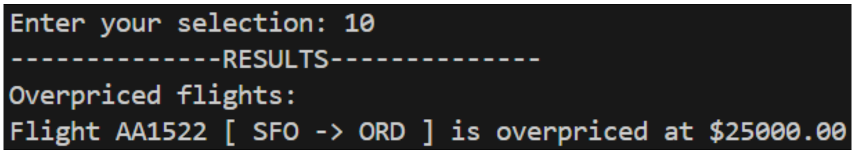


```sql
UPDATE Fare
SET Amount = 25000
WHERE Flight_number = 'AA1522';
```

```
#  Time      Action
1  22:20:56  UPDATE Fare SET Amount = 25000 WHERE Flight_number = 'AA1522'
```

This screenshot shows the changing price of flight AA1522 from $250 to $25,000.

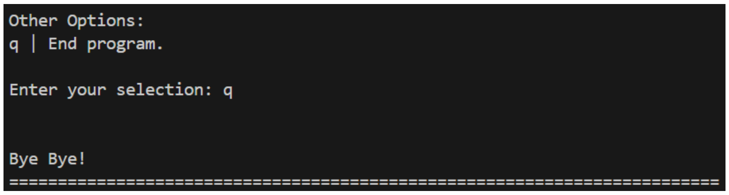
Output



```
Enter your selection: 10
--------------RESULTS--------------
Overpriced flights:
Flight AA1522 [ SFO -> ORD ] is overpriced at $25000.00
```

In this screenshot, the query is rerun to locate the new overpriced flight, which is now AA1522. This change to the flight's price is meant to test and show that the query works on varying data and not just on a provided dataset that remains static.

Exiting Screen



```
Other Options:
q | End program.

Enter your selection: q


Bye Bye!
==========================================================================
```

This final screenshot is meant to show that the program and connection to the database are closed smoothly without any issues.