This document contains a mathematical Mixed Integer Linear Programming (MILP) model and its corresponding code. Here's an overview of the report:

- The report is presented as a Jupyter notebook, with each code block of code is explained by markdown cells that explain its function.
- The formulations for the model are presented here, and following each formulation, the corresponding code is provided below it.
- Approximately 10-12 hours were invested in this project.
- Despite the effort, a feasible solution was not produced. The computation of decision variables takes about 20 minutes and then the - - - process crashes. Here are some strategies attempted to resolve this issue:
- - The Pulp solver, a free tool, is currently in use. One alternative could be to test the model with CPLEX or another commercial solver to see if that enhances performance. The decision variable X is programmed in different ways applicable to use CPLEX for demonstration.
- - Another approach is to refine the input data filtering process to capture only necessary information instead of the entire dataset.

I appologize for not being able to dedicate more time due to my existing workload.

Input Parameters:

Defined and used per as needed through the document

Decision Variables:

$\mathbf{x_{lat}}$: Binary decision variable it's 1 if aircraft *a* is assigned to fly leg *l* at time *t*, and 0 otherwise.

$\mathbf{y_{dat}}$: Binary decision variable. It takes a value of 1 if demand *d* is fulfilled by aircraft *a* at time *t*, and 0 otherwise.

$\mathbf{z_{la}}$: Binary decision variable. It takes a value of 1 if leg *l* is flown empty by aircraft *a*, and 0 otherwise.

$\mathbf{empty\_leg\_vars_{alt}}$: Binary decision variable. It takes a value of 1 if aircraft *a* flies leg *l* empty (without passengers) at time *t*, and 0 otherwise.

Import libraries

In [ ]:
```
from pulp import LpMaximize, LpProblem, LpVariable, lpSum, LpBinary, LpStatus
import pandas as pd
# import cplex
```

load data

In [ ]:
```
aircraft_data = pd.read_csv("C:/Users/mrkha/OneDrive/Desktop/OPTYM/code/Aircra
airports_data = pd.read_csv("C:/Users/mrkha/OneDrive/Desktop/OPTYM/code/Airpo
demands_file =  pd.read_csv("C:/Users/mrkha/OneDrive/Desktop/OPTYM/code/Deman
distance_time_file =  pd.read_csv("C:/Users/mrkha/OneDrive/Desktop/OPTYM/code/
```

Check for missing data

In [ ]:
```
aircraft_data.isnull().sum()
airports_data.isnull().sum()
demands_file.isnull().sum()
distance_time_file.isnull().sum()
```

Scenario 1D: Focus on a single day: 24th July

In [ ]:
```
specific_date = pd.Timestamp('2022-07-24')
demands_file['ScheduledDepDatetime'] = pd.to_datetime(demands_file['Scheduled
demands_file = demands_file[demands_file['ScheduledDepDatetime'].dt.date == s
```

Scenario 2D: Focus on a single day: 24th and 25th July

In [ ]:
```
# specific_date = [pd.Timestamp('2022-07-24'), pd.Timestamp('2022-07-25')]
# demands_file['ScheduledDepDatetime'] = pd.to_datetime(demands_file['Schedule
# demands_file = demands_file[demands_file['ScheduledDepDatetime'].dt.date.is
```

Preprocess

In [ ]:
```
aircraft_set = aircraft_data['AircraftID'].unique()
demands_set = demands_file['DemandID'].unique()
airports_set = airports_data['Airport Name'].unique()
L = list(set(zip(distance_time_file['DepAirport'], distance_time_file['ArrAir
# time_periods = [date.date() for date in specific_date] # for 2 day scenarios
time_periods = [specific_date.date()] # for 1 day scenarios
```

Parameters

In [ ]:
```
operating_cost_per_hour = 1500  # USD 1,500 per flying hour

operating_costs = {row['DepAirport'] + row['ArrAirport']: row['FlyingTime'] *
                   for _, row in distance_time_file.iterrows()}

revenue_per_hour = 5000  # USD 5,000 per flying hour

revenue_per_demand = {row['DemandID']: row['EstimatedBlocktime'] * revenue_pe
                      for _, row in demands_file.iterrows()}
```

In [ ]:
```
# Create a dictionary mapping each flight leg to its estimated block time (if
demand_block_time = {(row['DepAirport'], row['ArrAirport']): row['EstimatedBl
                     for _, row in demands_file.iterrows() if not pd.isna(row

# Create a dictionary mapping each flight leg to its flying time
flying_time = {(row['DepAirport'], row['ArrAirport']): row['FlyingTime']
               for _, row in distance_time_file.iterrows()}
```

In [ ]:
```
for l in L:
    dep_airport, arr_airport = l

    if l in demand_block_time:
        travel_time = demand_block_time[l]
    else:
        travel_time = flying_time.get(l, 0)  # Default to 0 if not found

        if travel_time > 4.5:
            travel_time += 1

    operating_costs[l] = travel_time
```

Model initialization

In [ ]:
```
model = LpProblem("Aircraft_Route_Optimization", LpMaximize)
# model = cplex.Cplex()
```

Decision Variables

In [ ]:
```
y_vars = LpVariable.dicts("y", [(d, a, t) for d in demands_set
                                for a in aircraft_set for t in time_periods],
```

In [ ]:
```
x_vars = LpVariable.dicts("x",
                          [(a, l, t) for a in aircraft_set for l in L for t in
                          cat=LpBinary)
```

In [ ]:
```
# # demonstration for testing with CPLEX
# x_var_names = ["x_" + "_".join([str(a), str(l), str(t)]) for a in aircraft_s

# # Add the binary variables to the CPLEX model
# model.variables.add(names=x_var_names, types=[model.variables.type.binary] *

# # Now 'x_vars' in the CPLEX model corresponds to the binary decision variabl
```

In [ ]:
```
z_vars = LpVariable.dicts("z",
                          [(a, l) for a in aircraft_set for l in L],
                          cat=LpBinary)
```

In [ ]:
```
empty_leg_vars = LpVariable.dicts("EmptyLeg",
                                  [(a, l, t) for a in aircraft_set for l in L
                                  cat=LpBinary)
```

Constraints

1. Demand Fulfillment: Each demand must be fulfilled at least once in the planning period.

$$\sum_t y_{dt} \geq 1 \quad \text{for all } d$$

In [ ]:
```
for d in demands_set:
    model += lpSum(y_vars[d, a, t] for a in aircraft_set for t in time_period
```

2. Aircraft Route Continuity: Ensures that for each aircraft, the number of arrivals at an airport equals the number of departures.

$$\sum_{l, S_{la}=k} X_{lat} = \sum_{l, E_{la}=k} X_{lat} \quad \text{for all } a, t, \text{ and } k \text{ in airports}$$

In [ ]:
```
for a in aircraft_set:
    for k in airports_set:
        for t in time_periods:
            arrivals = lpSum(x_vars[a, (dep, arr), t] for (dep, arr) in L if 
            departures = lpSum(x_vars[a, (dep, arr), t] for (dep, arr) in L i
            model += (arrivals == departures), f"Aircraft_Route_Continuity_{a
```

3. Aircraft Utilization Limit: Each aircraft cannot exceed its maximum hours of service.

$$\sum_{l} \sum_{t} x_{lat} \leq \text{HOS}_a \quad \text{for all } a$$

In [ ]:
```
for a in aircraft_set:
    for t in time_periods:
        total_hours_of_operation = lpSum(x_vars[a, l, t] for l in L)
        model += (total_hours_of_operation <= 12.5), f"Utilization_Limit_{a}_
```

4. Mandatory Rest: After reaching the maximum HOS, the aircraft must observe a mandatory rest period.

$$\begin{aligned} \text{If} \quad l \in L, \quad \sum x_{lat} &= \text{HOS}_a, \text{ then} \\ l \in L, \quad \sum x_{la(t+1)} &= 0 \end{aligned}$$

Given the complexity of directly implementing this as a linear constraint, a practical approach might be to set a utilization limit slightly less than the maximum HOS for each day, thereby implicitly allowing for rest time. This approach simplifies the model while achieving the intended outcome of ensuring rest periods.

In [ ]:
```
for a in aircraft_set:
    for t in range(len(time_periods) - 1):
        current_time = time_periods[t]
        next_time = time_periods[t + 1]
        time_difference = (next_time - current_time).total_seconds() / 3600

        if time_difference >= 12.5:
            model += (time_difference >= 22.5), f"Mandatory_Rest_{a}_{current_
```

5. Flight Leg Assignment: A flight leg can only be assigned if it is either loaded or flown empty.

$$x_{lat} \leq M \cdot z_{la} \quad \text{for all } l, a, \text{ and } t$$

In [ ]:
```
M = 10000
for a in aircraft_set:
    for l in L:
        for t in time_periods:
            model += x_vars[a, l, t] <= M * z_vars[a, l], f"Flight_Leg_Assignm
```

6. Ensures that each flight leg l is assigned to at most one aircraft

$$\sum_{a \in A} \sum_{t \in T} x_{alt} \leq 1 \quad \forall l \in L$$

In [ ]:
```
for l in L:
    model += lpSum(x_vars[a, l, t] for a in aircraft_set for t in time_period
```

7. Ensures the passenger capacity of an aircraft is not exceeded for each aircraft and each time period

$$\sum_{d \in D:(d,a,t) \in y\_vars} RevenuePerDemand_d \cdot y_{dat} \leq MaxPax_a \quad \forall a \in A, \forall t \in T$$

In [ ]:
```
for a in aircraft_set:
    for t in time_periods:
        max_capacity = aircraft_data.loc[aircraft_data['AircraftID'] == a, 'Ma
        assigned_passengers = lpSum(revenue_per_demand[d] * y_vars[d,a,t] for
        model += assigned_passengers <= max_capacity, f"Passenger_Capacity_{a}
```

8. Ensures an aircraft is available for its next flight only after accounting for the required turn-

$$\sum_{l \in L} \sum_{t_{prev} \in T : t_{prev} < t} x_{al_{t_{prev}}} + TurnAroundTime_a \leq \sum_{l \in L} x_{al_{t+1}} \quad \forall a \in A, \forall t \in T \setminus \{last\}$$

In [ ]:
```
for a in aircraft_set:
    for t in time_periods[:-1]:
        turn_around_time = aircraft_data.loc[aircraft_data['AircraftID'] == a
        available_time = lpSum(x_vars[a, l, t_prev] for l in L for t_prev in 
        model += available_time <= lpSum(x_vars[a, l, t] for l in L)
```

9. Ensures that each aircraft is limited to flying no more than one route

$$\sum_{l \in L} \sum_{t \in T} x_{alt} \leq 1 \quad \forall a \in A$$

In [ ]:
```
for a in aircraft_set:
    model += lpSum(x_vars[a, l, t] for l in L for t in time_periods) <= 1, f"
```

10. Ensure that each demand is assigned to at most one aircraft

$$\sum_{a \in A} \sum_{t \in T} y_{dat} \leq 1 \quad \forall d \in D$$

In [ ]:
```
for d in demands_set:
    model += lpSum(y_vars[d, a, t] for a in aircraft_set for t in time_period
```

11. Ensure that all demands are satisfied

$$\sum_{a \in A} \sum_{t \in T} y_{dat} \geq 1 \quad \forall d \in D$$

In [ ]:
```
for d in demands_set:
    model += lpSum(y_vars[d, a, t] for a in aircraft_set for t in time_period
```

12. Ensures that aircraft departure time aligns with scheduled departure datetime of demands

$$x_{adt} \leq y_{dat} \quad \forall d \in D, \forall a \in A, \forall t \in T$$

In [ ]:
```
for d in demands_set:
    for a in aircraft_set:
        for t in time_periods:
            model += x_vars[a, d, t] <= y_vars[d, a, t], f"Departure_Sync_{a}_
```

13. Each aircraft starts from its initial location

$$\sum_{dest \in \text{airports}:(init(a),dest) \in L} x_{a(init(a),dest)t_0} = 1 \quad \forall a \in A$$

In [ ]:
```
for a in aircraft_set:
    initial_location = aircraft_data.loc[aircraft_data['AircraftID'] == a, 'In
    model += lpSum(x_vars[a, (initial_location, dest), time_periods[0]]
                   for dest in airports_set if (initial_location, dest) in L)
```

14. Ensure that the total scheduled hours do not exceed the Initial HOS for each aircraft

$$\sum_{(dep,arr) \in L} \sum_{t \in T} OperatingCost_{(dep,arr)} \cdot x_{a(dep,arr)t} \leq InitialHOS_a \quad \forall a \in A$$

In [ ]:
```
for a in aircraft_set:
    initial_hos = aircraft_data.loc[aircraft_data['AircraftID'] == a, 'Initial
    model += lpSum(operating_costs[(dep, arr)] * x_vars[a, (dep, arr), t]
                   for (dep, arr) in L for t in time_periods) <= initial_hos,
```

16. This constraint defines the conditions under which a flight leg is considered empty. If the aircraft is flying the leg but not serving any demands, this difference will be positive, and the empty_leg_vars will be forced to 1, indicating an empty leg.

$$empty\_leg_{alt} \geq x_{alt} - \sum_{d \in D:d=l} y_{dat} \quad \forall a \in A, \forall l \in L, \forall t \in T$$

In [ ]:
```
for a in aircraft_set:
    for l in L:
        for t in time_periods:
            model += empty_leg_vars[a, l, t] >= x_vars[a, l, t] - lpSum(y_vars
```

objective function

Maximize total profit, which is the revenue from fulfilling demands minus the operating costs:

$$\text{Maximize } Z = \sum_{d \in D} \sum_{a \in A} \sum_{t \in T} RevenuePerDemand_d \cdot y_{dat} - W_2 * \sum_{l \in L} \sum_{a \in A} \sum_{t \in T} operating_{co}$$

Based on 3 cases for each scenarios, Ws can be modified to meet the requirement.

a) Focus of optimization search on maximization of total profit: in this case just make all Ws equal(=1) to focus on profit

b) Focus of optimization search on minimization of empty flying movements: increase the W3 to make sure it is avoiding he empty fliying movement to extend possible

c) Focus of optimization search on minimization of aircraft days used to satisfy the given demands: increasing the W2 to reduce the operational cost by having less days.

In [ ]:
```
revenue = lpSum(revenue_per_demand[d] * y_vars[d, a, t] for d in demands_set

operational_cost = lpSum(operating_costs[l] * x_vars[a, l, t] for a in aircra

empty_leg_penalty = 100000
empty_leg_cost = lpSum(empty_leg_penalty * empty_leg_vars[a, l, t] for a in a

model += revenue - operational_cost - empty_leg_cost, "Total_Profit"
```

solve the model

Now for each scenario, Ws must be modified before running the model

In [ ]:
```
model.solve()

if LpStatus[model.status] == 'Optimal':
    print("Optimal Solution Found!")
    for var in model.variables():
        if var.varValue is not None and var.varValue > 0:
            print(var.name, "=", var.varValue)
else:
    print("No optimal solution found. Status:", LpStatus[model.status])

# model.writeLP("model.lp")
```

output

After generationg the feasible solution and verify the validity of the solution, then next step is to format the output to meet the output schema.