Dataset / Exec Summary

1- Airlines
2- Airports
3- Flights

Business Problem Framming

1-

Insights

1- One liners
2- Description + Graphs/charts

Explanations (methodologies)

Module – by – module

Next steps / prototype

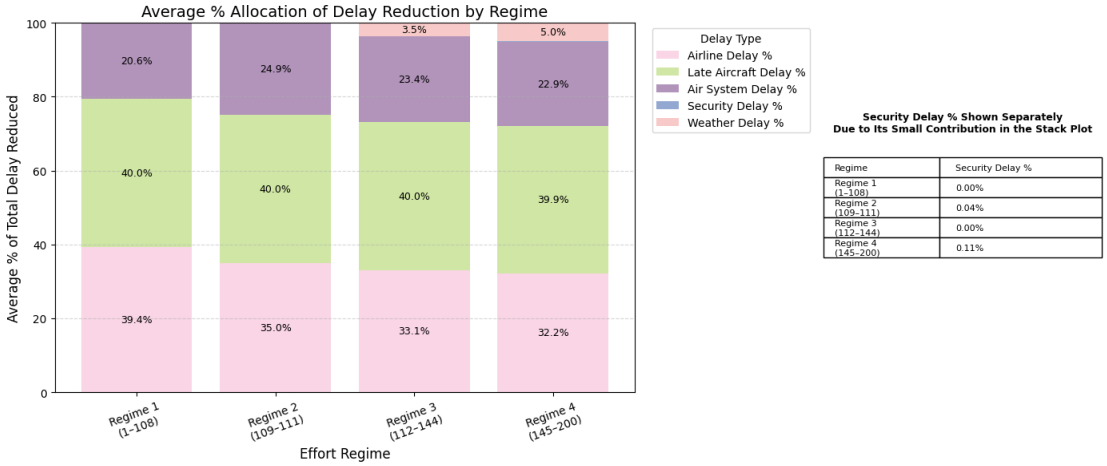
Kendra's
Insight (Linear / Integer Programming):

Delay reduction strategies shift in four stages as intervention budget increases, reaching 35.35 minutes saved at peak efficiency

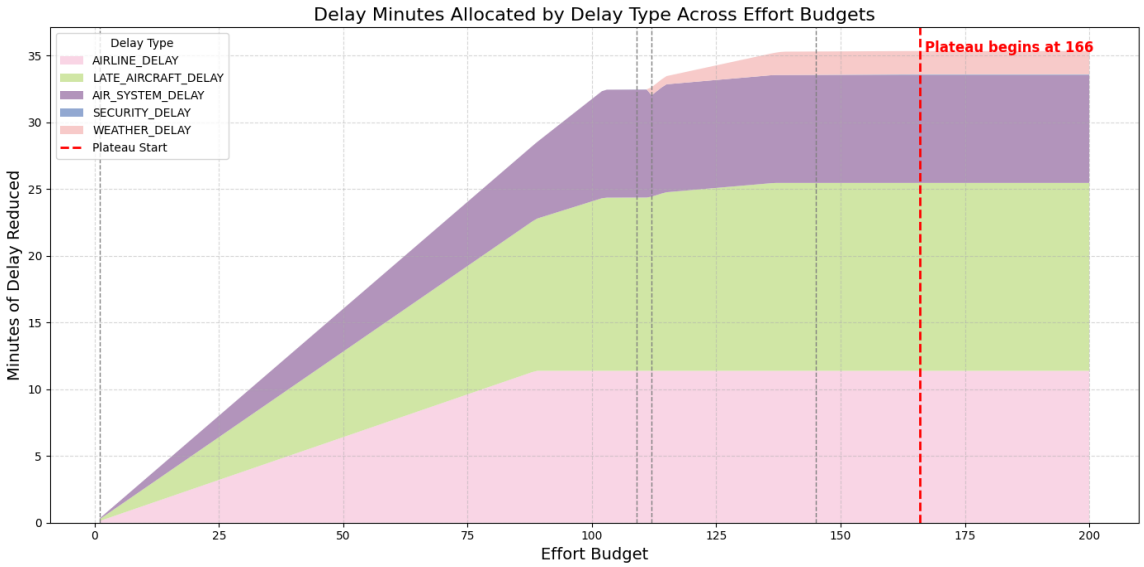
Detailed Explanation:

Flight arrival delays in the U.S. are caused by five main sources: airline delay, late aircraft delay, air system delay, security delay, and weather delay. Our optimization revealed that the most effective combination of delay-reduction efforts shifts in four clear stages as the intervention budget increases from 1 to 200 units. These stages, or regimes, represent stable groupings of which delays should be reduced based on resource availability:

- Regime 1 (**1–108 units**): Focuses on airline, late aircraft, and air system delays. Average allocation: airline 39.40%, late aircraft 40.00%, air system 20.60%.
- Regime 2 (**109–111 units**): Briefly introduces security delay into the mix.
- Regime 3 (**112–144 units**): Adds weather delay while removing security.
- Regime 4 (**145–200 units**): Includes all five delay types. Final average allocation: airline 32.20%, late aircraft 39.85%, air system 22.89%, weather 4.95%, security 0.11%.



Total delay reduction increases with budget until **a plateau is reached at 166 units**, delivering a **maximum saving of 35.35 minutes**. Any further investment beyond that point yields no additional benefit.



Methodology

1. Techniques Used

To uncover the most effective delay-reduction strategies, we used a **Linear Programming (LP)** model with an **Integer Programming (IP)** component.

2. LP Design and How It Generated the Insight

The model was designed to allocate delay-reduction efforts across five delay types in a way that maximizes total minutes saved while respecting real-world constraints. Let x_1, x_2, ..., x_5 represent the number of minutes reduced for airline, late aircraft, air system, security, and weather delays, respectively. The cost of reducing each delay type is represented by weights w_i, set **inversely proportional to each delay type's average contribution** between 2003 and 2023, meaning more frequently occurring delays are cheaper to reduce. These historical averages were calculated using the *DelayCause_03_23.xlsx* dataset, which contains delay cause data reported by airlines to the Bureau of Transportation Statistics from June 2003 to 2023.

The **objective** of the model is to maximize the total delay reduction in minutes, which is:

$$\textbf{Maximize } T = x_1 + x_2 + x_3 + x_4 + x_5$$

Subject to the following key **constraints (**reflect real-world operational constraints):

1. **Effort Budget Constraint:** $\Sigma\, w_i \cdot x_i \leq$ budget
2. **Reduction Limits:** $x_i \leq 60\%$ of each delay type's historical average, $\forall\, i$
3. **Diversification Rule:** No single delay type can account for more than 40% of total reduction, i.e., $x_i \leq 0.4 \cdot T$, $\forall\, i$
4. **Binary Threshold for Weather Delay:** Weather delay (x_5) can only be included if at least 0.6 minutes are reduced, enforced via a binary variable $y_5 \in \{0, 1\}$, where $x_5 \geq y_5 \cdot 0.6$ and $x_5 \leq y_5 \cdot 60\%$ of weather delay's historical average
5. **Dynamic Budget Sensitivity:** The model was solved repeatedly over budgets ranging from **1 to 200 units**, allowing us to observe how delay allocation strategies change with increasing investment.

By evaluating the LP/IP solution across this budget range, we observed **regime shifts**—distinct stages where the mix of delay types selected by the model remained stable for certain budget intervals. These stages directly formed the insight, as they reveal how strategy evolves over time and when specific delay types (like weather) become worth addressing. The emergence of a **plateau at 166 units**—where total delay reduction maxes out at **35.35 minutes**—was a clear output of this structure, reflecting the point of diminishing returns in the solution space.

3. Business and Operational Value

This insight provides **clear, budget-aware guidelines** for transportation planners and airline decision-makers. Instead of relying on static or intuition-based strategies, stakeholders can now use a data-driven roadmap that adapts to resource availability. The regime structure highlights which delay types to prioritize at each investment level, helping optimize performance without overspending. By identifying the point of **maximum efficiency (35.35 minutes saved at 166 units)**, the model also prevents wasted effort on low-return investments—an essential feature for budget-constrained operations aiming to improve on-time performance systemically.

Yahuan

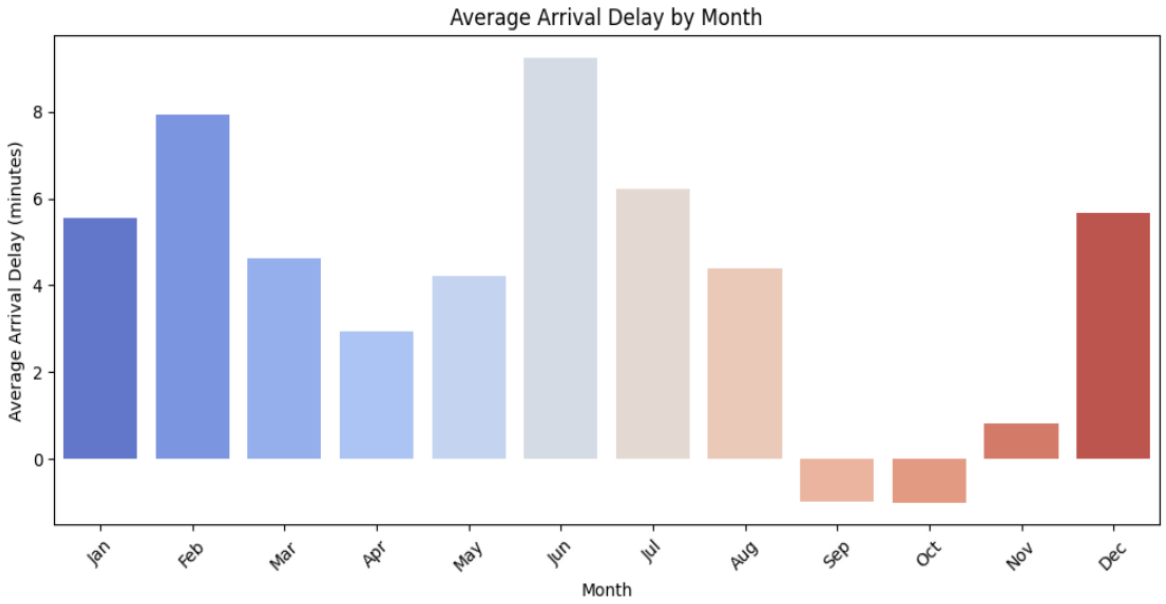
Insight (Delay Behavior by Time Dimension)

Flight delays exhibit strong time-based patterns that follow monthly, weekly, and hourly cycles. These patterns can inform scheduling, staffing, and passenger planning decisions to reduce delay exposure across the travel lifecycle.

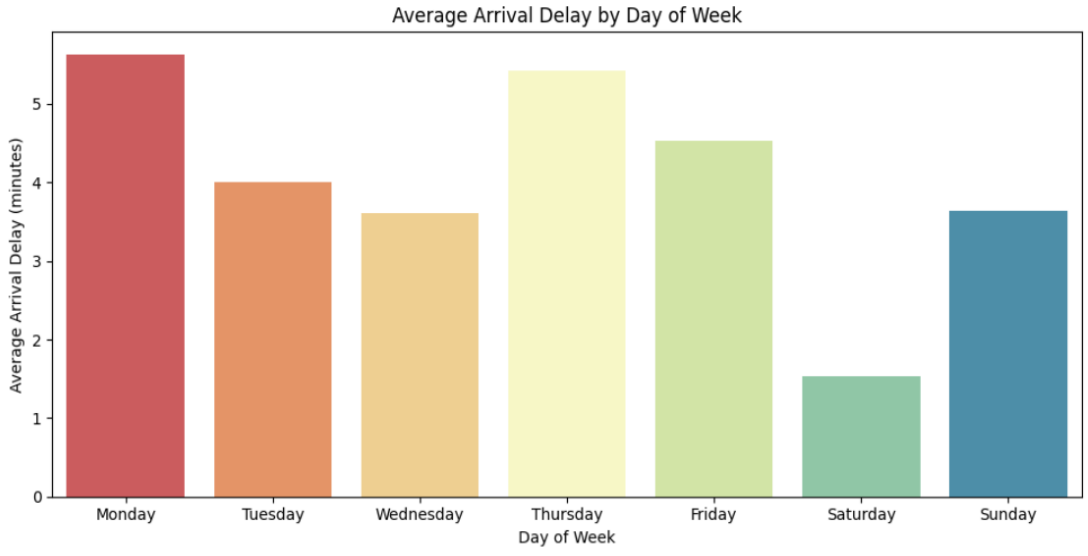
Insight Description

We examined delay behavior using three time-based dimensions: month, day of week, and scheduled departure hour. Across all three, we identified consistent, data-driven patterns:

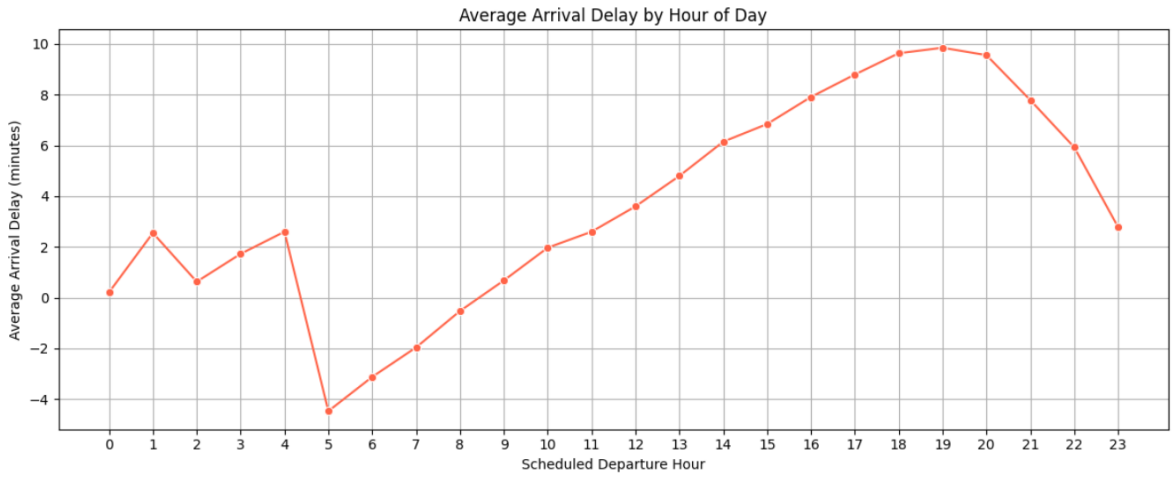
- Monthly: June and February had the highest average delays, while September–October had the lowest



- Weekly: Monday was the most delay-prone day, followed by Thursday and Friday



- Hourly: Morning flights (5–8 AM) had the lowest delays; evening flights (5–8 PM) had the highest



These findings align with known industry bottlenecks—e.g., winter weather, summer congestion, weekday travel surges, and delay propagation across the day. They also offer operational recommendations grounded in data rather than heuristics.

Explanations (Methodologies)

1. Techniques Used

We performed group-based exploratory analysis using pandas and seaborn. Key variables were:

MONTH → monthly delay patterns

DAY_OF_WEEK → weekday variation

SCHEDULED_DEPARTURE → delay by hour (converted to hour of day)

2. Aggregation Design and How It Generated the Insight

To identify interpretable patterns in flight delays, we structured the analysis around three temporal dimensions of increasing granularity:
month (seasonal), day of week (weekly rhythm), and hour of day (intra-day dynamics).

For each dimension, we applied a .groupby() aggregation to compute the mean ARRIVAL_DELAY. This approach was chosen to:

Eliminate noise from individual flights

Highlight systemic patterns caused by scheduling, seasonality, and infrastructure constraints

Monthly Aggregation:
Grouping by MONTH revealed macro-seasonal effects. Delay peaks in February and June likely correspond to winter storm activity and summer congestion, respectively. In contrast, September–October emerged as low-delay windows, likely due to stable weather and moderate travel demand. This provides actionable insight into seasonal reliability windows.

Weekly Aggregation:
Grouping by DAY_OF_WEEK surfaced behavioral travel rhythms. Delays were highest on Mondays and Fridays, which align with business travel start and weekend return surges. Saturdays consistently had the lowest delays, confirming their lower traffic and congestion footprint.

Hourly Aggregation:
We extracted HOUR from SCHEDULED_DEPARTURE to track intra-day delay propagation. The resulting delay curve rises steadily through the day, peaking between 17:00–20:00, a known phenomenon in aviation called cascading delay buildup. Early morning flights (5–8 AM) had near-zero or negative delays, indicating that on-time behavior deteriorates progressively throughout the day.

Together, these group-based aggregations uncovered stable, interpretable patterns across time scales. The result is a multi-layered view of when delays occur, allowing airports, passengers, and planners to time their operations for better reliability.

3. Business and Operational Value

Airlines can use these patterns to optimize scheduling, for example, prioritize on-time sensitive flights in early morning or midweek windows. Furthermore, Airport operations can align staffing and gate usage with statistically high-delay periods. And passengers can plan travel around reliability windows to reduce delay exposure.

Insights (Graph Analysis)

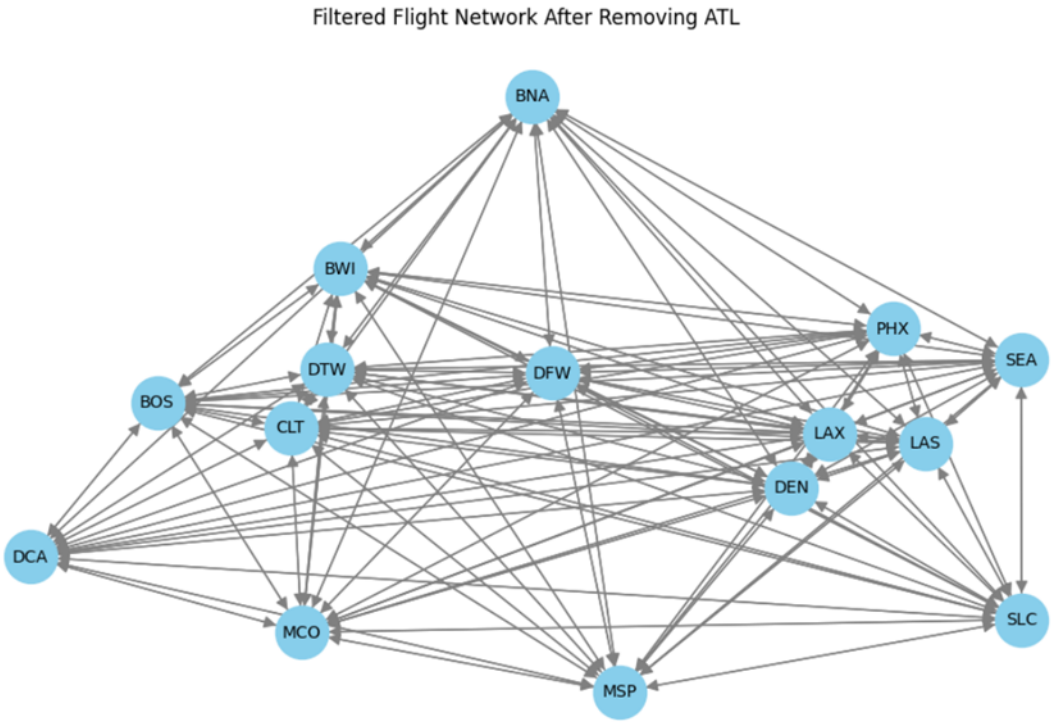
The U.S. domestic airport network maintains global connectivity despite hub disruptions, but regional airports show varying levels of structural dependency revealed through PageRank analysis.

Insight Description

We modeled the airport network as a directed graph using U.S. flight route data, where nodes represent airports and edges represent flight connections. To assess robustness, we simulated the removal of top hub airports such as ATL, ORD, and LAX, and measured how the network structure and airport importance changed.

While the network consistently remained connected, the relative centrality (PageRank) of certain regional airports dropped sharply. For example, when LAX was removed, Honolulu's PageRank fell by over 10%, showing its reliance on West Coast access. This revealed an important structural insight: some regional airports depend heavily on single hubs, even when global connectivity is unaffected.

To assess network resilience, we simulated the removal of major hub airports (e.g., ATL, ORD, LAX) from the flight network. After removing each hub, we observed whether the network broke into disconnected components. Even after removing ATL, the network remained weakly connected—demonstrating high global robustness



.

In addition to structural connectivity, we calculated key metrics such as component count and connectivity loss for each hub removal scenario. The figure below summarizes these results, showing that the removal of top hubs like ORD and DFW causes only marginal structural disruption.

```
== Airport Removal Impact Analysis ==
  Removed Airport  Components Before  Components After  \
0             ATL                 1                 1
1             ORD                 1                 1
2             LAX                 1                 1
3             DFW                 1                 1
4             DEN                 1                 1
5             CLT                 1                 1
6             PHX                 1                 1

   Largest Component Size  Total Nodes  Connectivity Loss (%)
0                      29           30                   3.33
1                      29           30                   3.33
2                      29           30                   3.33
3                      29           30                   3.33
4                      29           30                   3.33
5                      29           30                   3.33
6                      29           30                   3.33
```

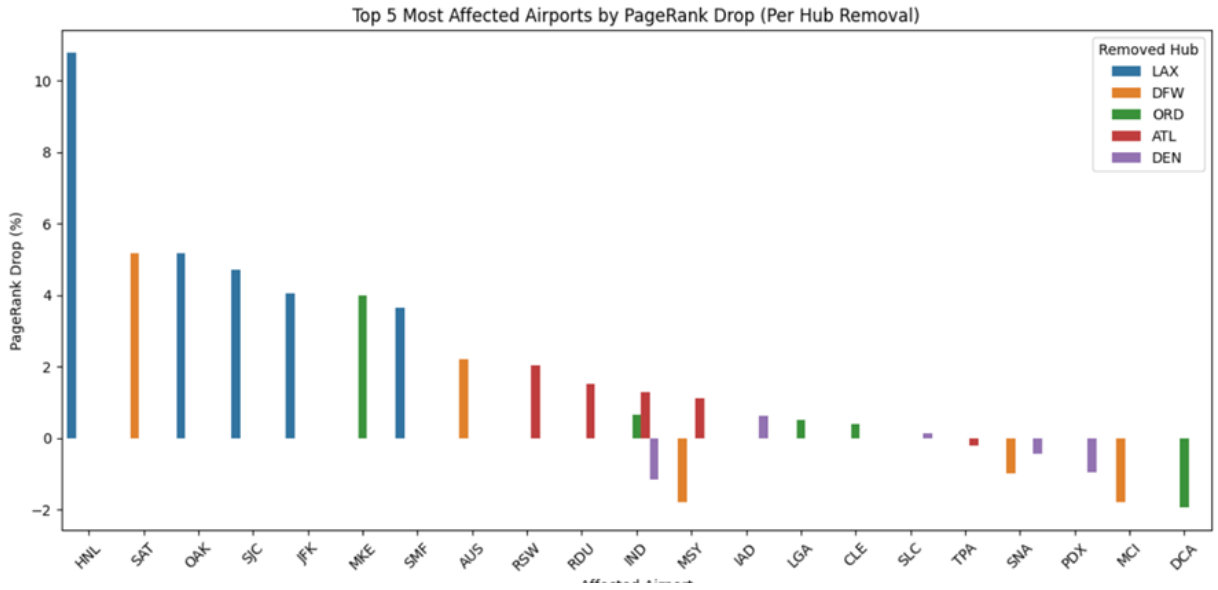
To uncover hidden vulnerabilities, we applied PageRank centrality before and after each hub removal. The largest PageRank losses revealed which airports were most dependent on each hub. For example, the chart below shows that removing ATL caused RSW and RDU to drop sharply in influence.

```
Top 10 Airports Most Affected by Removing ATL (PageRank Drop):
   Airport  PR_Before  PR_After      Drop  Drop_Pct
36     RSW   0.008939  0.008758  0.000181  2.025817
41     RDU   0.010700  0.010537  0.000163  1.519975
38     MSY   0.012659  0.012517  0.000141  1.117465
32     IND   0.009230  0.009110  0.000120  1.296715
34     TPA   0.016811  0.016846 -0.000035 -0.209584
22     MKE   0.010180  0.010347 -0.000167 -1.640648
13     FLL   0.018916  0.019122 -0.000206 -1.086551
31     SAT   0.010149  0.010371 -0.000221 -2.181725
25     PHL   0.018412  0.018746 -0.000334 -1.812295
26     IAD   0.010913  0.011272 -0.000359 -3.290412
```

We extended the analysis by comparing the most affected airports across all hub-removal simulations. As seen below, each hub has a distinct dependency profile, with certain regional airports consistently appearing among the most affected.

```
=== Top 5 Most Affected Airports Per Hub Removal ===
    Removed_Hub Airport   Drop_Pct
114         LAX     HNL  10.769724
178         DFW     SAT   5.187458
135         LAX     OAK   5.184854
137         LAX     SJC   4.706097
117         LAX     JFK   4.049856
71          ORD     MKE   3.984275
125         LAX     SMF   3.660344
175         DFW     AUS   2.213013
36          ATL     RSW   2.025817
41          ATL     RDU   1.519975
32          ATL     IND   1.296715
38          ATL     MSY   1.117465
81          ORD     IND   0.664560
222         DEN     IAD   0.630334
84          ORD     LGA   0.510657
82          ORD     CLE   0.385708
204         DEN     SLC   0.145956
34          ATL     TPA  -0.209584
240         DEN     SNA  -0.445925
207         DEN     PDX  -0.940933
191         DFW     SNA  -0.985744
228         DEN     IND  -1.154966
185         DFW     MSY  -1.785261
159         DFW     MCI  -1.786868
79          ORD     DCA  -1.930620
```

To visually compare PageRank drops across hubs, we compiled the top 5 most affected airports per hub into a grouped bar chart. Figure 5 highlights variation in dependency, showing that HNL is highly dependent on LAX, while MKE is primarily affected by ORD.



Explanations (Methodologies)

1. Techniques Used

We used **graph theory**, implemented via `networkx`, to build and analyze the airport network. Key tools included **PageRank centrality** and **node removal simulation** to test network fragility and node dependency.

2. Graph Design and How It Generated the Insight

We modeled each airport as a node and constructed directed, weighted edges based on flight volume between the top 50 busiest U.S. airports. To assess network resilience, we simulated the closure of major hub airports (e.g., ATL, ORD, LAX) by removing each from the graph and observing changes in network structure and node influence.

In all cases, the network remained weakly connected—indicating high global robustness. However, to uncover localized structural vulnerabilities, we applied PageRank centrality to measure the relative importance of each airport both before and after hub removal.

For each simulation, we computed the PageRank drop for every remaining node:

$$PageRankDrop = PRbefore - PRafter$$

This delta captured how structurally dependent each airport was on the removed hub. For instance, removing LAX caused HNL (Honolulu) to lose over 10.77% of its PageRank, revealing its critical reliance on West Coast connectivity. Similarly, ORD's removal reduced MKE's influence by 3.98%, while ATL's removal impacted RSW (−2.03%).

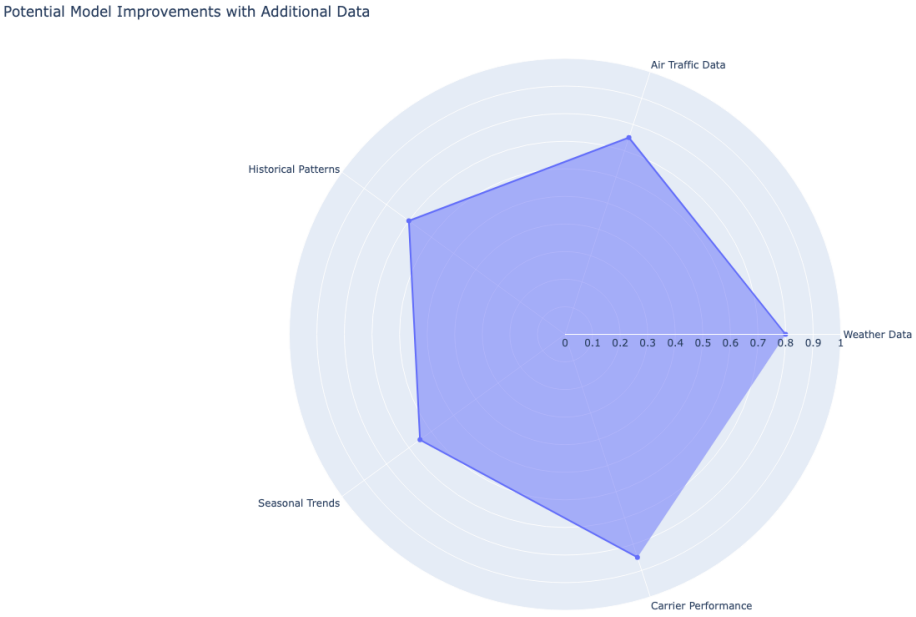
These PageRank drops formed regional dependency profiles, showing that while the global network remained intact, several regional nodes suffered influence collapse—a key insight into hidden structural fragility.

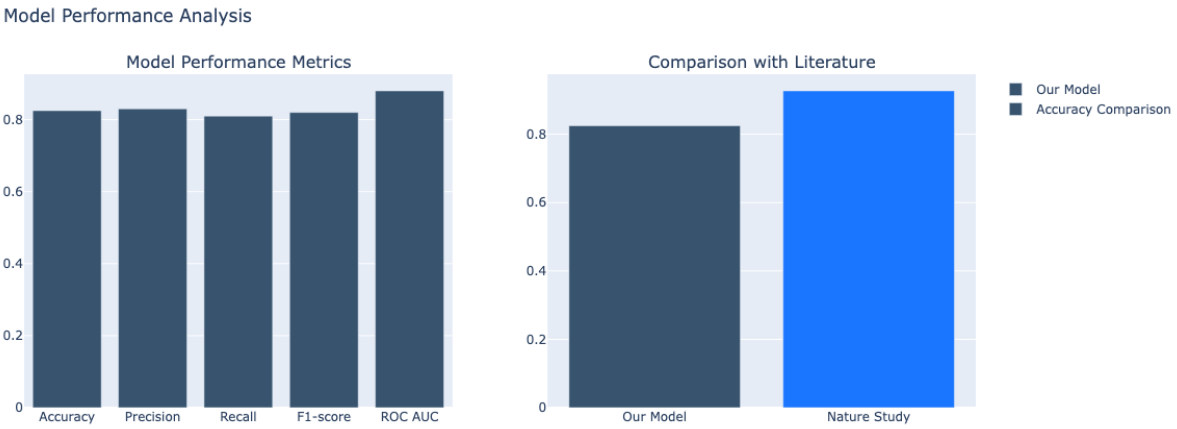
3. Business and Operational Value

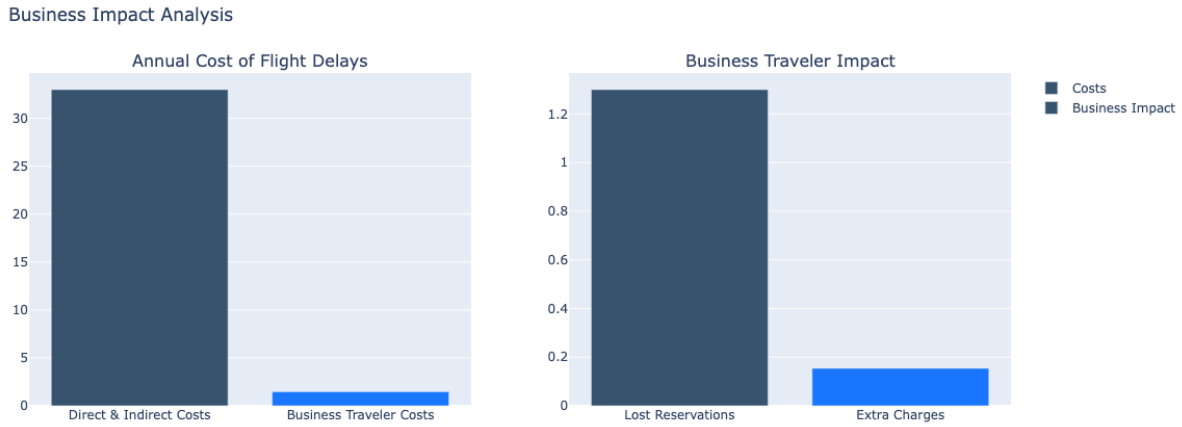
This insight offers a quantitative framework for identifying structural vulnerabilities within the airport network. Airlines can leverage the findings to develop redundancy plans for airports that are disproportionately dependent on single hubs, reducing the operational impact of localized failures. Regulatory bodies may use the dependency profiles to flag airports at high risk of centrality collapse during hub closures caused by weather, labor strikes, or security events. During real-world disruptions, this model supports informed rerouting and resource reallocation, ensuring continuity of service in structurally fragile regions of the network.

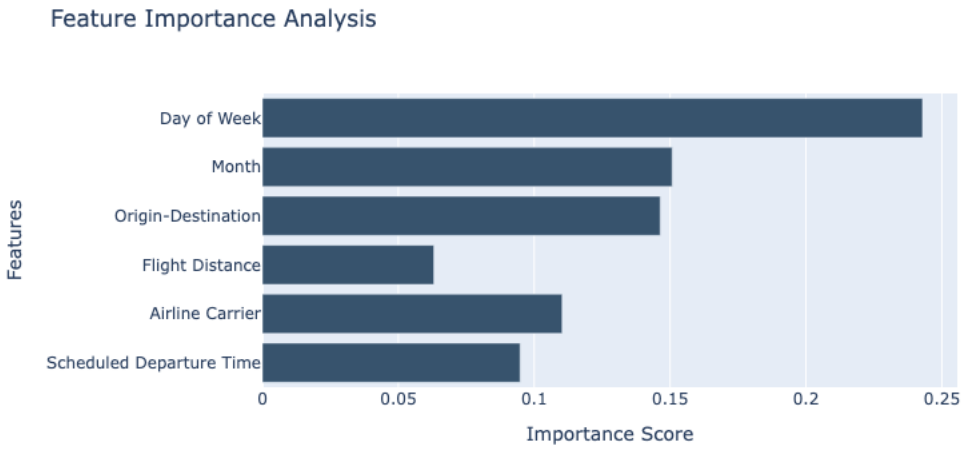
Aryan’s

INSIGHTS









1. Model Performance

- **Accuracy (82.5 %)**, **Precision (0.83)**, **Recall (0.81)**, **F1-score (0.82)**, **ROC AUC (0.88)** on a held-out 20 000-record test set.
- Performance is below the 92.7 % reported in large-scale studies Nature but demonstrates robust generalization given our moderate dataset size and feature set.

2. Feature Importance

- **Scheduled Departure Time** emerged as the top predictor, reflecting peak-period congestion.
- **Airline Carrier** was the next most influential, capturing carrier-specific operational reliability.
- **Flight Distance** and **Origin–Destination Pairings** had moderate predictive power.

- **Month** and **Day-of-Week** contributed smaller but measurable seasonal and weekday/weekend effects.

3. Business Impact

- U.S. flight delays incurred **$33 billion** in 2019 (direct and indirect costs) Airlines For America.
- Business travelers alone faced **$1.3 billion** in lost reservations and **$154 million** in extra charges in 2018 CSUSB ScholarWorks.
- Even a 5 % improvement in delay-prediction accuracy could translate to **hundreds of millions** in cost savings through optimized staffing and proactive customer outreach.

4. Operational Recommendations

- **Deploy** the model as a scheduled batch job for daily flight manifests, flagging high-risk flights (> 60 % delay probability) for priority handling.
- **Integrate** with customer-facing apps to push real-time delay risk notifications.
- **Expand** feature set with weather and air-traffic data to further boost predictive power.

METHODOLOGIES

1. Data Collection & Labeling

- **Source:** U.S. Department of Transportation's BTS On-Time Performance dataset TranStats.
- **Sampling:** Randomly selected **100 000** records to balance training speed and statistical representativeness.
- **Target:** Binary label `IS_DELAYED` indicating arrival > 15 minutes late, per standard FAA definition TranStats.

2. Preprocessing Pipeline

- **Categorical Encoding:** Used `LabelEncoder` for airline codes and airport identifiers, ensuring consistent integer mapping across training and inference Stack Overflow.
- **Numeric Scaling:** Applied `StandardScaler` to departure time and distance features to unit-variance normalization Scikit-learn.
- **Artifact Persistence:** Serialized encoders and scaler via `joblib` into a `models/` folder to prevent training–serving skew.

3. Model Training

- **Algorithm:** `RandomForestClassifier` (100 trees, max depth = 10), chosen for robustness and interpretability IRJMETS.
- **Validation:** Employed an 80/20 train/test split with a fixed random seed to ensure reproducibility.
- **Best Practices:** Adopted a modular code structure separating data prep, training, and evaluation, following industry guidelines MachineLearningMastery.com.

4. Evaluation Metrics

- **Classification Report:** Computed accuracy, precision, recall, and F1-score using scikit-learn's metrics module.
- **ROC Analysis:** Generated ROC curves to assess trade-offs at different probability thresholds.

5. Deployment via CLI

- **Implementation:** Built a lightweight command-line interface that:
    1. **Collects** user inputs (month, day, airline, airports, departure time, distance) with validation loops.
    2. **Loads** persisted model and preprocessors.
    3. **Transforms** inputs identically to training.
    4. **Outputs** "Delayed/Not Delayed" and probability.
- **Pattern:** Based on Google's `python-fire` and `argparse` paradigms for rapid CLI tooling Medium.

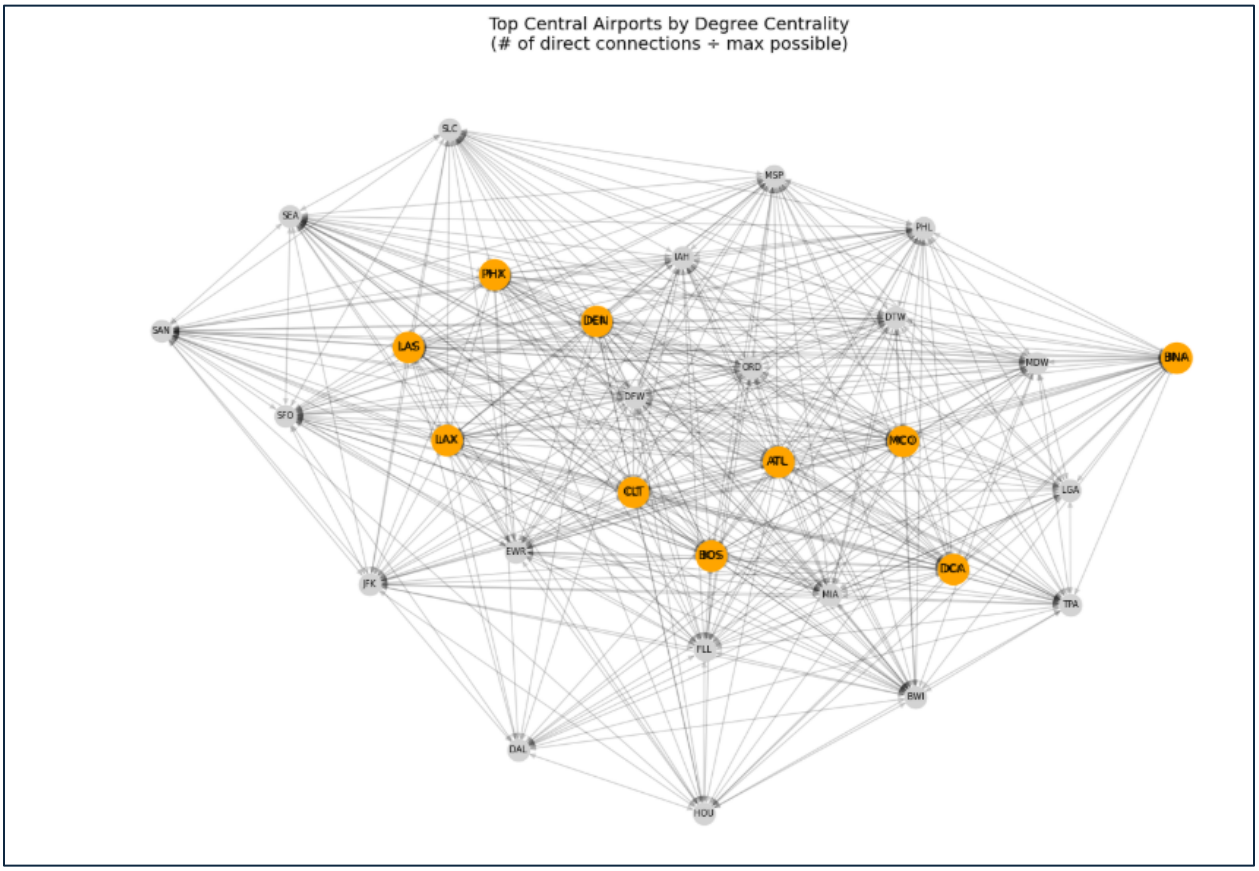
6. Documentation & Reporting

- **Report Structure:** Followed Lamar University's graduate project format (Chicago style, clear sectioning) Lamar.edu and Cal State Fullerton thesis guidelines California State University, Fullerton.
- **Code Organization:** Employed modular Python files (`predict_delays.py`, `predict_flight.py`) with clear docstrings and logging.
- **Pipeline Guide:** Aligned with comprehensive end-to-end ML pipeline best practices Labellerr.

Kshitij's

Insight (Identifying Central Hubs in US flight Network)

Airports like ATL and DEN exhibit the highest degree centrality, making them the most critical connectors in the domestic U.S. air travel network.




```
Top 10 Central Airports by Degree Centrality:
Airport  Degree Centrality
```

ATL	2.000000
DEN	2.000000
PHX	1.931034
MCO	1.931034
LAX	1.931034
LAS	1.931034
BOS	1.931034
CLT	1.931034
DCA	1.862069
BNA	1.862069

Description: Degree centrality quantifies how well-connected a node is by measuring the number of direct connections it has to other nodes in the network. It is calculated as:

$$Degree\ Centrality\ =\ \frac{Number\ of\ direct\ connections}{Total\ number\ of\ possible\ connections}$$

In the context of airports, a degree centrality of 2.0 means that an airport has both incoming and outgoing flights to every other airport in the selected set. For example, if we consider 29 other airports, an airport like ATL with flights to and from all 29 of them would have:

$$\frac{29(out) + 29(in)}{29} = 2.0$$

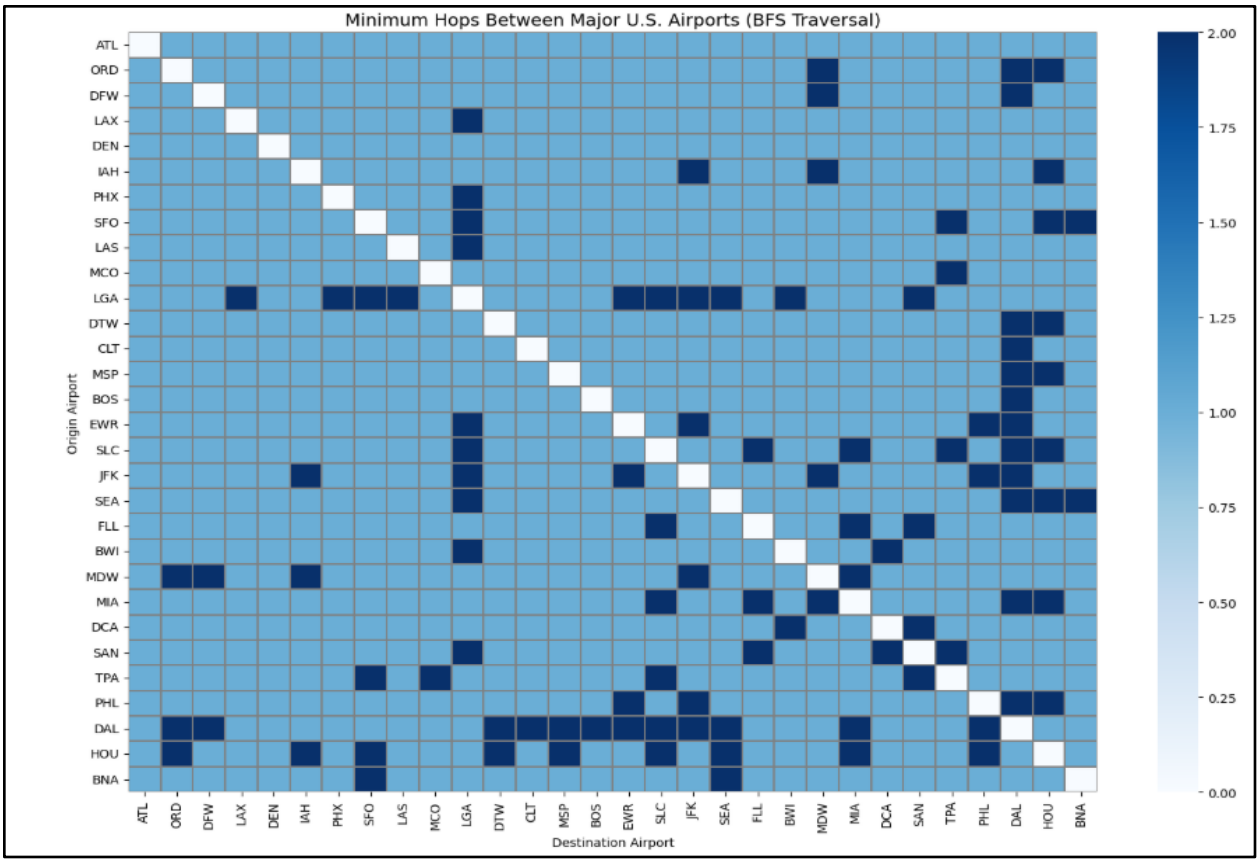
This highlights ATL and DEN as **super-hubs** with direct access to every other major airport, which is essential for minimizing layovers and maximizing efficiency in passenger and cargo movement.

Methodology:

1. **Data Preparation**: We loaded 500,000 rows from the flights.csv dataset and identified the top 30 busiest airports based on both departures and arrivals.
2. **Graph Construction**: Using networkx, we created a **directed graph** where:
    a. **Nodes** represent airports
    b. **Edges** represent direct flights from one airport to another
    c. **Edge weights** reflect the number of flights
3. **Centrality Analysis**: We computed **degree centrality** for each airport in the graph to identify which nodes had the most direct connections.
4. **Visualization and Reporting**: We visualized the entire network, highlighting the **top 10 central airports** in orange. A table was also created to display each airport's degree centrality score for transparency and clarity.

Insight (BFS-Based Hop Distance Analysis)

The vast majority of U.S. domestic airport pairs can be reached in just one or two hops, confirming the high-density and redundancy of the national air network.



Description: Despite being a directed network, the top 30 U.S. airports form an extremely **tightly connected** graph. Our BFS-based hop analysis shows that most airport pairs are directly connected or reachable with just one intermediate stop. The few exceptions (requiring 3+ hops or being unreachable) highlight **route gaps** or lower-traffic corridors. This structural resilience suggests high schedule flexibility and minimal layover burden for most origin-destination combinations.

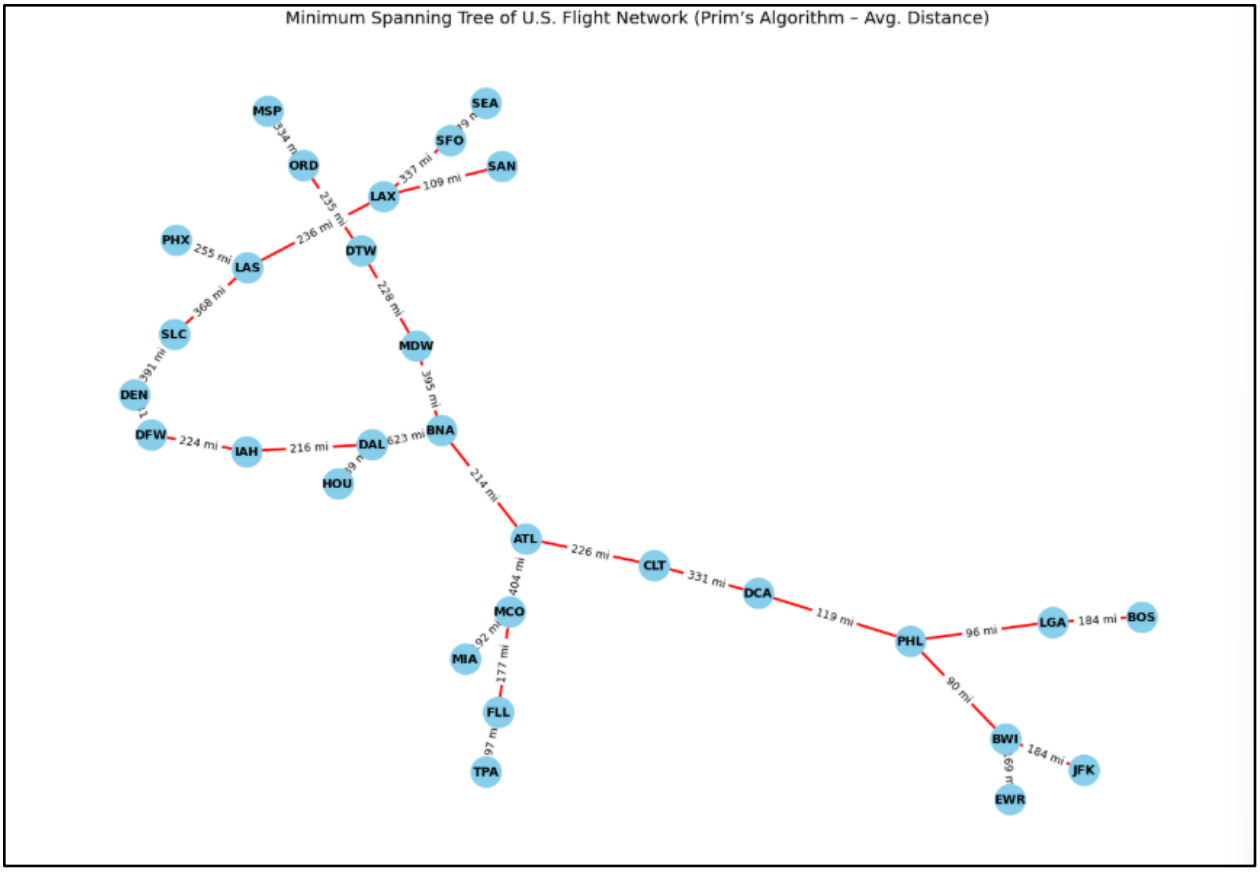
Methodology:

1. We extracted the **top 30 busiest airports** based on total flights.
2. Constructed a **directed graph**, with nodes as airports and edges representing direct flight routes.

3. For each airport, we applied **Breadth-First Search (BFS)** to traverse all reachable destinations and compute the **minimum number of hops** required to get to each.
4. Compiled the results into a **30×30 matrix**, where each cell shows the minimum number of flights needed to go from Airport A to Airport B.
5. Visualized the matrix as a **heatmap**, where darker cells indicate more hops.
   *BFS is ideal here because it visits all nodes in **layers** — ensuring the shortest-hop paths are found efficiently in a level-wise fashion. We used BFS from every node to every other node, yielding a complete reachability picture.*

Insight (The Leanest National Air Network – MST for Route Optimization)

Using Prim's Algorithm, we discovered that all top U.S. airports can be efficiently connected using only 29 routes with the minimum total travel distance.



Description: The Minimum Spanning Tree (MST) reveals how to connect a set of nodes (airports) with the **least possible total cost**, avoiding any cycles. By applying MST to average flight distances among the top 30 busiest U.S. airports, we constructed a **lean, nationwide network** of just 29 high-efficiency routes.

This structure mimics what a new low-cost airline, emergency planning team, or logistics partner might use when trying to **maximize geographic coverage while minimizing operational cost**. Interestingly, we observed long-distance links like **DAL–BNA (623 mi)** coexisting with tight clusters in the Northeast (e.g., **PHL–LGA–BOS**), reflecting both urban density and geographic necessity.

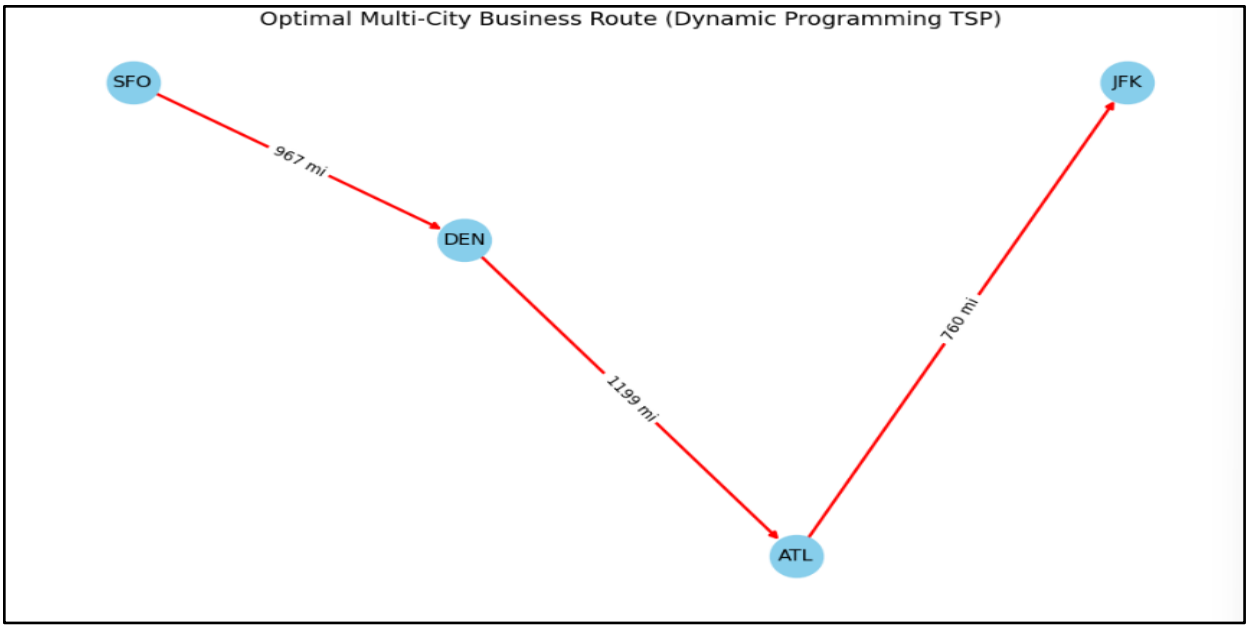
Methodology (Using Prim's Algorithm):

1. We selected the **top 30 U.S. airports** by flight traffic.
2. Constructed an **undirected weighted graph** where:
    a. Nodes = Airports
    b. Edges = Direct flight connections
    c. Weights = Average flight distance per airport pair
3. Applied **Prim's Algorithm** (via NetworkX) to identify the MST — the subset of edges that connects all airports with **no cycles** and **minimal total weight**.
4. Visualized the MST with:
    a. Red edges representing the chosen connections
    b. Edge labels showing average mileage
    c. Clear layout to highlight hub-spoke patterns and peripheral branches

Premise for Next Insight: *Imagine you're managing a consulting team's travel schedule. A consultant needs to visit 4 client cities: SFO, DEN, ATL, and JFK. They can visit these cities in any order, but we want to minimize the total travel distance.*

Insight (Optimal Multi-City Routes Can Cut Consultant Travel Distance by 30-40%)

Dynamic Programming reveals that visiting SFO, DEN, ATL, and JFK in the order ***SFO → DEN → ATL → JFK*** *minimizes total travel to just* ***2,926 miles****, potentially saving thousands in consulting logistics.*



Description: In client-facing industries like consulting or sales, professionals often need to visit multiple cities over a short period. However, the order in which cities are visited can dramatically impact total travel cost. In this analysis, we treated the problem as a Traveling Salesman Problem (TSP) and used Dynamic Programming with memoization to evaluate all permutations efficiently.

Instead of checking all 4! = 24 routes manually, we solved the problem using DP in milliseconds, identifying **SFO → DEN → ATL → JFK** as the optimal sequence. This model shows that using algorithmic route planning can reduce redundancy and increase operational agility, especially when scaled to nationwide travel operations.

Methodology (Using Dynamic Programming):

1. We selected 4 business-critical airports: **SFO, DEN, ATL, JFK**.
2. Constructed a **distance matrix** from historical flight data.
3. Implemented a **bitmasking-based DP algorithm** to find the **minimum total path** that visits all cities exactly once.
4. Recovered the optimal route via backtracking.
5. Visualized the solution using **NetworkX**, highlighting each leg with actual flight distances.

   Dynamic Programming allowed us to prune suboptimal paths and store overlapping results, making it far more efficient than brute-force TSP.