|  |  |
| --- | --- |
| **A black and red logo  AI-generated content may be incorrect.** | **CS 112 Computer Programming for Engineering**  **Date: July 7, 2025**  **Due Date: August 2, 2025** |

**Course Final Project**

**Title: Integrated Data Science and Software Engineering Project: Global Aviation Network Analysis and CanvasLMS-Lite Development**

**Introduction**

In this ambitious and interdisciplinary course project, students will collaboratively tackle two real-world inspired challenges by combining the rigor of data science with the creativity of software engineering. The project is designed for a team of four students, each contributing unique strengths in programming, analytics, user interface design, and systems thinking.

The project has two core components, each targeting critical and practical skills needed in today’s technology-driven world: **Global Aviation Network Analysis** and **CanvasLMS-Lite Development**.

**Global Aviation Network Analysis: A Comprehensive Data Science Project**

The first component focuses on applying data science techniques to study and gain insights from global aviation data. Air transportation is one of the most complex and globally interconnected systems, impacting economic development, international trade, and human mobility.

Students will collect, clean, and process large datasets containing information about flights, airports, airlines, and routes. Using Python and popular data science libraries (such as Pandas, NumPy, and NetworkX), the team will perform exploratory data analysis (EDA), network modeling, and statistical interpretation to uncover patterns in air traffic flows, identify key hubs, and analyze how factors like geography, seasonality, or external disruptions (e.g., pandemics or weather conditions) affect the global aviation network.

Students will visualize these insights using advanced plotting tools (Matplotlib, Seaborn, Plotly) and present their findings through clear, reproducible Jupyter Notebooks and dashboards. The goal is to demonstrate practical mastery of data wrangling, network science, and insight communication.

**CanvasLMS-Lite: A Scalable, GUI-Based Learning Management System in Python**

The second component challenges the team to design and build a **lightweight Learning Management System (LMS)** inspired by modern platforms like Canvas LMS but tailored as a simplified prototype.

This project requires students to integrate principles of software engineering, user-centered design, and Python programming to deliver a functional GUI-based application. The system should allow instructors to create and manage courses, enroll students, upload resources, and track basic progress. Students will design a clear, intuitive Graphical User Interface using Tkinter or PyQt and implement essential features such as user authentication, role-based access, and a scalable database backend (SQLite or MySQL).

The team will produce clean, modular, well-documented code that follows best practices for maintainability and extensibility. Additionally, students must consider scalability—designing the system architecture in a way that could, in principle, be expanded for larger deployments.

**Bringing It All Together**

This integrated project bridges theory and application. By completing both parts, students will demonstrate proficiency in:

* **Data acquisition and preprocessing**
* **Statistical and network analysis**
* **Data visualization and insight communication**
* **Full-cycle software development**
* **Database design and integration**
* **GUI design and usability testing**
* **Version control and collaborative coding**

Working in a team of four, students will divide roles and responsibilities strategically—for example, two members may lead the aviation network analysis while the other two focus on the LMS development, with frequent cross-team updates to encourage interdisciplinary learning.

In the final deliverable, students will submit a detailed project report, clean and reproducible code repositories, well-commented Jupyter Notebooks, and a working demo of the CanvasLMS-Lite application.

Through this project, students will not only reinforce their technical skills but also gain valuable experience in managing complex, multifaceted tasks—preparing them for careers in data science, software engineering, and systems integration.

**Global Aviation Network Analysis: A Comprehensive Data Science Project**

**Project Overview**

This project involves analyzing three interconnected aviation datasets (airlines, airports, and routes) to understand global aviation patterns, network structures, and business intelligence opportunities. Students will work in teams of four over a 3-week period to clean, analyze, and visualize aviation data to derive meaningful insights about the global aviation industry.

**Introduction**

The aviation industry represents one of the world's most complex transportation networks, connecting thousands of airports across hundreds of countries through millions of flights annually. This project provides students with real-world datasets containing information about airlines, airports, and flight routes to explore fundamental questions about network analysis, geographic patterns, business intelligence, and data visualization.

Through this project, students will gain hands-on experience with data cleaning, exploratory data analysis, network theory, geospatial analysis, and interactive visualization techniques while working with authentic industry data. The project emphasizes collaborative work, combining technical skills with analytical thinking to solve real-world problems in the aviation sector.

**Learning Objectives**

By the end of this project, students will be able to:

**Technical Skills**

* **Data Cleaning & Preprocessing**: Clean messy real-world datasets, handle missing values, standardize formats, and join multiple datasets
* **Exploratory Data Analysis**: Perform comprehensive statistical analysis and identify patterns in large datasets
* **Network Analysis**: Apply graph theory concepts to analyze transportation networks and identify key nodes and connections
* **Geospatial Analysis**: Work with geographic data, create interactive maps, and analyze spatial patterns
* **Data Visualization**: Create compelling visualizations including interactive dashboards, network graphs, and geographic maps
* **Programming Proficiency**: Develop advanced Python skills using pandas, numpy, matplotlib, seaborn, plotly, networkx, and folium

**Analytical Skills**

* **Business Intelligence**: Extract actionable insights from data to inform business decisions
* **Pattern Recognition**: Identify trends, anomalies, and relationships in complex datasets
* **Statistical Analysis**: Apply appropriate statistical methods to validate findings
* **Critical Thinking**: Evaluate data quality, methodology limitations, and interpretation validity

**Collaboration & Communication**

* **Team Management**: Effectively distribute tasks and coordinate work across team members
* **Documentation**: Create clear, reproducible code and comprehensive project documentation
* **Presentation Skills**: Communicate findings effectively to both technical and non-technical audiences
* **Peer Review**: Provide constructive feedback and incorporate suggestions from team members

**Part A: Detailed Project Tasks**

**Week 1: Data Foundation and Exploration**

**Task 1.1: Data Cleaning and Preprocessing (All Team Members)**

**Objective**: Transform raw CSV files into clean, usable datasets

**Specific Activities**:

# Step 1: Load and examine raw data

import pandas as pd

import numpy as np

# Load datasets with proper headers

airlines\_cols = ['airline\_id', 'name', 'alias', 'iata', 'icao', 'callsign', 'country', 'active']

airports\_cols = ['airport\_id', 'name', 'city', 'country', 'iata', 'icao', 'latitude', 'longitude', 'altitude', 'timezone', 'dst', 'tz\_name', 'type', 'source']

routes\_cols = ['airline', 'airline\_id', 'source\_airport', 'source\_airport\_id', 'dest\_airport', 'dest\_airport\_id', 'codeshare', 'stops', 'equipment']

# Step 2: Handle missing values

# Replace '\N' with NaN

# Decide on imputation strategies for different columns

# Document decisions and rationale

# Step 3: Data validation

# Verify airport codes exist in airports dataset

# Check for duplicate entries

# Validate geographic coordinates

# Ensure data type consistency

**Deliverables**:

* Three clean CSV files with proper headers
* Data cleaning report documenting all transformations
* Basic statistics summary for each dataset
* Data quality assessment with identified issues

**Task 1.2: Exploratory Data Analysis (Team Member 2 - Data Analyst)**

**Objective**: Understand dataset characteristics and identify initial patterns

**Specific Activities**:

* Generate descriptive statistics for all numerical variables
* Create frequency distributions for categorical variables
* Identify top airlines by number of routes
* Find busiest airports by number of connections
* Analyze geographic distribution of airports and routes
* Examine airline activity status and country distribution

**Deliverables**:

* Comprehensive EDA report with visualizations
* Summary statistics tables
* Initial hypothesis about network structure
* List of interesting patterns for further investigation

**Task 1.3: Data Integration and Relationship Mapping (Team Member 1 - Data Engineer)**

**Objective**: Successfully join datasets and establish relationships

**Specific Activities**:

# Create master dataset by joining all three tables

# Establish foreign key relationships

# Handle orphaned records (routes without corresponding airlines/airports)

# Create lookup dictionaries for efficient querying

# Validate join operations and document data loss

**Deliverables**:

* Integrated dataset with all relevant information
* Entity relationship diagram
* Data dictionary with all fields explained
* Join operation documentation and validation results

**Week 2: Advanced Analysis and Insights**

**Task 2.1: Network Analysis (Team Member 1 - Data Engineer)**

**Objective**: Analyze the aviation network as a complex graph structure

**Specific Activities**:

import networkx as nx

import matplotlib.pyplot as plt

# Create network graph

G = nx.Graph()

# Add airports as nodes with attributes (country, coordinates, etc.)

# Add routes as edges with weights (frequency, distance, etc.)

# Calculate network metrics

# - Node centrality measures (degree, betweenness, closeness, PageRank)

# - Network diameter and average path length

# - Clustering coefficients

# - Community detection

# - Hub identification

# Analyze network structure

# - Identify superhubs (airports with highest connectivity)

# - Find bridge routes (critical connections)

# - Detect isolated components

# - Measure network efficiency

**Specific Analyses**:

* **Hub Analysis**: Identify the most connected airports globally and by region
* **Centrality Analysis**: Find airports that serve as critical intermediaries
* **Community Detection**: Identify regional clusters and airline alliance patterns
* **Path Analysis**: Calculate shortest paths between major cities
* **Network Resilience**: Assess impact of removing major hubs

**Deliverables**:

* Network analysis report with key findings
* Interactive network visualization
* Hub ranking and classification
* Network metrics dashboard

**Task 2.2: Geographic and Geospatial Analysis (Team Member 3 - Visualization Specialist)**

**Objective**: Analyze spatial patterns and geographic distribution

**Specific Activities**:

import folium

import geopandas as gpd

from geopy.distance import geodesic

# Geographic analysis

# - Calculate route distances using haversine formula

# - Analyze airport density by region/country

# - Identify geographic clusters and gaps

# - Map airline route networks

# - Analyze seasonal patterns (if temporal data available)

# Spatial visualizations

# - World map with all airports colored by traffic volume

# - Route density heatmaps

# - Airline-specific route maps

# - Regional connectivity analysis

**Specific Analyses**:

* **Regional Connectivity**: Compare aviation development across continents
* **Distance Analysis**: Categorize routes by distance (short/medium/long haul)
* **Geographic Gaps**: Identify underserved regions
* **Airline Territory**: Map each airline's geographic footprint
* **Airport Clustering**: Find geographic clusters of high-traffic airports

**Deliverables**:

* Interactive world map with multiple layers
* Regional analysis report
* Distance distribution analysis
* Geographic clustering visualization

**Task 2.3: Business Intelligence and Market Analysis (Team Member 2 - Data Analyst)**

**Objective**: Extract business insights for strategic decision-making

**Specific Activities**:

# Market analysis

# - Airline market share by routes and regions

# - Competition analysis on popular routes

# - Identify market gaps and opportunities

# - Calculate route profitability proxies

# - Analyze airline operational strategies

# Business metrics

# - Route frequency and capacity utilization

# - Airline fleet composition analysis

# - Market concentration indices

# - Competitive landscape mapping

**Specific Analyses**:

* **Market Share Analysis**: Dominant airlines by region and route type
* **Competition Mapping**: Routes with multiple airlines and monopolistic routes
* **Growth Opportunities**: Underserved city pairs with high potential
* **Airline Strategies**: Compare network structures of different airline types
* **Route Profitability**: Proxy analysis using distance, competition, and market size

**Deliverables**:

* Business intelligence dashboard
* Market analysis report
* Competitive landscape visualization
* Strategic recommendations document

**Week 3: Integration and Presentation**

**Task 3.1: Comprehensive Dashboard Development (All Team Members)**

**Objective**: Create an interactive dashboard integrating all analyses

**Specific Activities**:

# Using Streamlit or Dash

import streamlit as st

import plotly.express as px

import plotly.graph\_objects as go

# Dashboard components:

# - Executive summary with key metrics

# - Interactive world map with filtering options

# - Network analysis visualization

# - Business intelligence charts

# - Search functionality for specific routes/airlines

# - Comparison tools for different metrics

**Dashboard Features**:

* **Overview Tab**: Key statistics and executive summary
* **Network Tab**: Interactive network graphs with filtering
* **Geography Tab**: World map with multiple overlays
* **Business Tab**: Market analysis and competitive intelligence
* **Search Tab**: Route finder and airline comparison tools

**Deliverables**:

* Fully functional interactive dashboard
* User guide for dashboard navigation
* Technical documentation for dashboard maintenance

**Task 3.2: Advanced Visualizations and Insights (Team Member 3 - Visualization Specialist)**

**Objective**: Create compelling visualizations that tell the story of global aviation

**Specific Activities**:

# Create publication-quality visualizations

# - Animated route maps showing temporal patterns

# - 3D network visualizations

# - Interactive chord diagrams for route flows

# - Heatmaps for route density and competition

# - Comparative charts for airline performance

**Visualization Types**:

* **Network Graphs**: Force-directed layouts showing airport connections
* **Chord Diagrams**: Flow between regions and countries
* **Heatmaps**: Route density and competition intensity
* **Animated Maps**: Growth patterns over time (if temporal data available)
* **Comparative Charts**: Airline and airport performance metrics

**Deliverables**:

* Gallery of high-quality visualizations
* Interactive visualization showcase
* Design documentation and style guide

**Task 3.3: Documentation and Presentation (All Team Members)**

**Objective**: Create comprehensive documentation and presentation materials

**Specific Activities**:

* Write technical documentation for all code
* Create executive summary of key findings
* Develop presentation slides for stakeholders
* Record video demonstration of dashboard
* Prepare poster for project showcase

**Deliverables**:

* Complete technical documentation
* Executive summary report
* Presentation slides (20-30 slides)
* Video demonstration (10-15 minutes)
* Project poster for academic presentation

**Technical Implementation Guide**

**Required Libraries and Tools**

# Data manipulation and analysis

import pandas as pd

import numpy as np

import scipy.stats as stats

# Network analysis

import networkx as nx

# Visualization

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import plotly.graph\_objects as go

import plotly.figure\_factory as ff

# Geographic analysis

import folium

import geopandas as gpd

from geopy.distance import geodesic

# Dashboard development

import streamlit as st

# or

import dash

from dash import dcc, html, Input, Output

# Utilities

import warnings

warnings.filterwarnings('ignore')

**Data Cleaning Templates**

# Template for handling missing values

def clean\_aviation\_data(df, missing\_value\_indicators=['\N', 'NULL', '', ' ']):

"""

Clean aviation dataset by handling missing values and standardizing formats

"""

# Replace missing value indicators with NaN

df = df.replace(missing\_value\_indicators, np.nan)

# Handle specific data types

# Convert coordinates to float

# Standardize country names

# Clean airport/airline codes

return df

# Template for data validation

def validate\_relationships(airlines\_df, airports\_df, routes\_df):

"""

Validate relationships between datasets

"""

# Check for orphaned routes

# Validate airport codes

# Ensure airline consistency

return validation\_report

**Analysis Templates**

# Network analysis template

def analyze\_network\_structure(routes\_df, airports\_df):

"""

Perform comprehensive network analysis

"""

G = nx.Graph()

# Add nodes (airports)

for idx, airport in airports\_df.iterrows():

G.add\_node(airport['iata'], \*\*airport.to\_dict())

# Add edges (routes)

for idx, route in routes\_df.iterrows():

if route['source\_airport'] in G.nodes and route['dest\_airport'] in G.nodes:

G.add\_edge(route['source\_airport'], route['dest\_airport'])

# Calculate metrics

centrality = nx.degree\_centrality(G)

betweenness = nx.betweenness\_centrality(G)

return G, centrality, betweenness

# Geographic analysis template

def create\_route\_map(routes\_df, airports\_df):

"""

Create interactive map of airline routes

"""

m = folium.Map(location=[0, 0], zoom\_start=2)

# Add airports

for idx, airport in airports\_df.iterrows():

folium.CircleMarker(

location=[airport['latitude'], airport['longitude']],

popup=airport['name'],

radius=5

).add\_to(m)

# Add routes

for idx, route in routes\_df.iterrows():

# Get coordinates and draw line

pass

return m

**Part B: Detailed Project Tasks (More Related Tasks)**

### Task 1: Loading and Inspecting the Datasets

**Objective**: Load the CSV files into Python, assign correct column names, and inspect their structure and content.

**Why?** Understanding the structure of your data is the first step in any data analytics project. This task introduces students to pandas and basic data inspection.

**Python Code**:

import pandas as pd  
  
# Define column names for each dataset  
airlines\_columns = ['Airline ID', 'Name', 'Alias', 'IATA', 'ICAO', 'Callsign', 'Country', 'Active']  
airports\_columns = ['Airport ID', 'Name', 'City', 'Country', 'IATA', 'ICAO', 'Latitude', 'Longitude',   
 'Altitude', 'Timezone', 'DST', 'Tz database timezone', 'Type', 'Source']  
routes\_columns = ['Airline', 'Airline ID', 'Source airport', 'Source airport ID',   
 'Destination airport', 'Destination airport ID', 'Codeshare', 'Stops', 'Equipment']  
  
# Load datasets with correct headers and handle '\N' as NaN  
airlines = pd.read\_csv('airlines.csv', names=airlines\_columns, na\_values='\\N')  
airports = pd.read\_csv('airports.csv', names=airports\_columns, na\_values='\\N')  
routes = pd.read\_csv('routes.csv', names=routes\_columns, na\_values='\\N')  
  
# Inspect the data  
print("Airlines DataFrame Info:")  
print(airlines.info(), "\n")  
print("Airlines First 5 Rows:")  
print(airlines.head(), "\n")  
  
print("Airports DataFrame Info:")  
print(airports.info(), "\n")  
print("Airports First 5 Rows:")  
print(airports.head(), "\n")  
  
print("Routes DataFrame Info:")  
print(routes.info(), "\n")  
print("Routes First 5 Rows:")  
print(routes.head(), "\n")

**Explanation**: - pd.read\_csv loads the CSV files into pandas DataFrames. - The names parameter assigns proper column names based on the OpenFlights dataset structure. - na\_values='\\N' converts \N (used in OpenFlights for missing data) to NaN. - .info() provides a summary of columns, data types, and non-null counts. - .head() displays the first five rows to preview the data.

**Expected Outcome**: - Console output showing the structure (columns, data types, non-null counts) and first five rows of each DataFrame. - Confirmation that the datasets are loaded correctly with appropriate column names.

**Student Task**: - Run the code and save the output (e.g., as a screenshot or text file). - Note any unexpected data types (e.g., Latitude as object instead of float) or high numbers of missing values.

### Task 2: Cleaning the Datasets

**Objective**: Handle missing values, correct data types, and remove duplicates to prepare the data for analysis.

**Why?** Real-world datasets often contain errors or missing data. Cleaning ensures accurate analysis and prevents errors in later steps.

**Python Code**:

# Check for missing values  
print("Missing Values in Airlines:")  
print(airlines.isnull().sum(), "\n")  
print("Missing Values in Airports:")  
print(airports.isnull().sum(), "\n")  
print("Missing Values in Routes:")  
print(routes.isnull().sum(), "\n")  
  
# Convert numeric columns to appropriate types  
airports['Latitude'] = pd.to\_numeric(airports['Latitude'], errors='coerce')  
airports['Longitude'] = pd.to\_numeric(airports['Longitude'], errors='coerce')  
airports['Altitude'] = pd.to\_numeric(airports['Altitude'], errors='coerce')  
airports['Timezone'] = pd.to\_numeric(airports['Timezone'], errors='coerce')  
  
# Check for duplicates  
print("Duplicate Rows in Airlines:", airlines.duplicated().sum())  
print("Duplicate Rows in Airports:", airports.duplicated().sum())  
print("Duplicate Rows in Routes:", routes.duplicated().sum())  
  
# Drop duplicates if any  
airlines = airlines.drop\_duplicates()  
airports = airports.drop\_duplicates()  
routes = routes.drop\_duplicates()  
  
# Handle missing values (example: fill missing IATA/ICAO with 'Unknown')  
airlines['IATA'] = airlines['IATA'].fillna('Unknown')  
airlines['ICAO'] = airlines['ICAO'].fillna('Unknown')  
airports['IATA'] = airports['IATA'].fillna('Unknown')  
airports['ICAO'] = airports['ICAO'].fillna('Unknown')  
routes['Codeshare'] = routes['Codeshare'].fillna('N') # Assume no codeshare if missing  
  
# Verify data types and missing values after cleaning  
print("\nAfter Cleaning - Airlines Info:")  
print(airlines.info(), "\n")  
print("After Cleaning - Airports Info:")  
print(airports.info(), "\n")  
print("After Cleaning - Routes Info:")  
print(routes.info(), "\n")

**Explanation**: - .isnull().sum() counts missing values in each column to identify problematic areas. - pd.to\_numeric converts columns like Latitude and Longitude to numeric types, with errors='coerce' turning invalid values into NaN. - .drop\_duplicates() removes duplicate rows, if any. - Missing IATA and ICAO codes are filled with ‘Unknown’ to preserve data for analysis. - Missing Codeshare values in routes are assumed to be ‘N’ (no codeshare).

**Expected Outcome**: - Output showing the number of missing values and duplicates before cleaning. - Confirmation that numeric columns have correct data types (e.g., float64 for Latitude). - Reduced missing values and no duplicates after cleaning.

**Student Task**: - Run the code and document the number of missing values and duplicates before and after cleaning. - If there are still many missing values (e.g., in Callsign or City), suggest a strategy to handle them (e.g., drop rows or fill with a placeholder).

### Task 3: Exploratory Data Analysis (EDA)

**Objective**: Analyze the datasets to uncover trends, such as the distribution of airlines by country, busiest airports, or common aircraft types.

**Why?** EDA helps identify patterns and insights, guiding further analysis and visualization.

**Python Code**:

import matplotlib.pyplot as plt  
  
# Airlines: Top 10 countries by number of airlines  
plt.figure(figsize=(10, 6))  
airlines['Country'].value\_counts().head(10).plot(kind='bar', title='Top 10 Countries by Number of Airlines')  
plt.xlabel('Country')  
plt.ylabel('Number of Airlines')  
plt.tight\_layout()  
plt.show()  
  
# Airports: Distribution of airports by country  
plt.figure(figsize=(10, 6))  
airports['Country'].value\_counts().head(10).plot(kind='bar', title='Top 10 Countries by Number of Airports')  
plt.xlabel('Country')  
plt.ylabel('Number of Airports')  
plt.tight\_layout()  
plt.show()  
  
# Routes: Top 10 source airports  
plt.figure(figsize=(10, 6))  
routes['Source airport'].value\_counts().head(10).plot(kind='bar', title='Top 10 Source Airports by Routes')  
plt.xlabel('Airport IATA Code')  
plt.ylabel('Number of Routes')  
plt.tight\_layout()  
plt.show()  
  
# Summary statistics for numeric columns in airports  
print("Airports Numeric Summary:")  
print(airports[['Latitude', 'Longitude', 'Altitude', 'Timezone']].describe(), "\n")  
  
# Count of active vs. inactive airlines  
print("Active Airlines Count:")  
print(airlines['Active'].value\_counts(), "\n")

**Explanation**: - .value\_counts() counts occurrences of unique values (e.g., countries or airports). - matplotlib creates bar charts to visualize the top 10 countries or airports. - .describe() provides summary statistics (mean, min, max) for numeric columns like Latitude and Altitude. - The Active column in airlines is analyzed to see how many airlines are currently operational.

**Expected Outcome**: - Bar charts showing: - Top 10 countries with the most airlines (likely USA, China, etc.). - Top 10 countries with the most airports. - Top 10 source airports by number of routes (e.g., ATL, LAX). - Summary statistics for airports numeric columns (e.g., range of altitudes). - Count of active (Y) vs. inactive (N) airlines.

**Student Task**: - Save the charts as PNG files using plt.savefig('filename.png'). - Write a brief summary of key findings (e.g., “The USA has the most airlines and airports”). - Suggest one additional EDA question (e.g., “What are the most common aircraft types in routes?”).

### Task 4: Merging and Advanced Analysis

**Objective**: Merge the datasets to answer complex questions, such as which airlines operate the most routes from specific airports or countries.

**Why?** Merging datasets allows you to combine information for richer insights, a key skill in data analytics.

**Python Code**:

# Merge routes with airports to get source airport details  
routes\_with\_source = routes.merge(airports[['Airport ID', 'Name', 'City', 'Country']],   
 left\_on='Source airport ID', right\_on='Airport ID',   
 how='left', suffixes=('', '\_source'))  
  
# Merge with destination airport details  
routes\_with\_airports = routes\_with\_source.merge(airports[['Airport ID', 'Name', 'City', 'Country']],   
 left\_on='Destination airport ID', right\_on='Airport ID',   
 how='left', suffixes=('\_source', '\_destination'))  
  
# Preview merged data  
print("Merged Routes DataFrame:")  
print(routes\_with\_airports[['Airline', 'Source airport', 'Name\_source', 'City\_source', 'Country\_source',   
 'Destination airport', 'Name\_destination', 'City\_destination', 'Country\_destination']].head(), "\n")  
  
# Analyze routes by airline and source country  
routes\_by\_airline\_country = routes\_with\_airports.groupby(['Airline', 'Country\_source']).size().reset\_index(name='Route Count')  
top\_routes = routes\_by\_airline\_country.sort\_values(by='Route Count', ascending=False).head(10)  
print("Top 10 Airlines by Route Count from Source Country:")  
print(top\_routes, "\n")  
  
# Visualize top routes  
plt.figure(figsize=(12, 6))  
top\_routes.plot(kind='bar', x='Airline', y='Route Count',   
 title='Top 10 Airlines by Route Count from Source Country')  
plt.xlabel('Airline')  
plt.ylabel('Number of Routes')  
plt.tight\_layout()  
plt.show()

**Explanation**: - .merge() combines routes with airports using Airport ID as the key. - how='left' ensures all routes are kept, even if some airport data is missing. - groupby and size() count routes by airline and source country. - The bar chart visualizes the top airlines by route count.

**Expected Outcome**: - A merged DataFrame showing routes with source and destination airport names, cities, and countries. - A table and chart of the top 10 airlines by route count, including their source country. - Insights like “Airline X operates the most routes from the USA.”

**Student Task**: - Run the code and save the merged DataFrame as a CSV (routes\_with\_airports.to\_csv('merged\_routes.csv')). - Identify one additional question to answer with the merged data (e.g., “Which country pairs have the most routes?”). - Document any issues (e.g., missing airport data after merging).

### Task 5: Visualization and Network Analysis

**Objective**: Create interactive visualizations and model the flight routes as a network to analyze connectivity.

**Why?** Visualizations make findings accessible, and network analysis reveals structural patterns in the data, such as key hub airports.

**Python Code**:

import plotly.express as px  
import networkx as nx  
  
# Interactive map of airports  
fig = px.scatter\_geo(airports, lat='Latitude', lon='Longitude', hover\_name='Name',   
 color='Country', title='Global Airport Locations',   
 projection='natural earth')  
fig.write\_layout(showlegend=False)  
fig.show()  
  
# Network analysis of routes  
G = nx.from\_pandas\_edgelist(routes, source='Source airport', target='Destination airport', create\_using=nx.DiGraph())  
print(f"Number of nodes (airports): {G.number\_of\_nodes()}")  
print(f"Number of edges (routes): {G.number\_of\_edges()}")  
  
# Calculate degree centrality (number of connections per airport)  
degree\_centrality = nx.degree\_centrality(G)  
top\_airports = sorted(degree\_centrality.items(), key=lambda x: x[1], reverse=True)[:10]  
print("Top 10 Airports by Degree Centrality:")  
for airport, centrality in top\_airports:  
 print(f"{airport}: {centrality:.4f}")  
  
# Visualize a small subset of the network (e.g., top 50 routes)  
subset\_routes = routes.head(50) # Limit for visualization  
G\_subset = nx.from\_pandas\_edgelist(subset\_routes, source='Source airport', target='Destination airport',   
 create\_using=nx.DiGraph())  
plt.figure(figsize=(12, 8))  
nx.draw(G\_subset, with\_labels=True, node\_size=500, node\_color='lightblue', font\_size=10, arrows=True)  
plt.title('Network of Top 50 Flight Routes')  
plt.show()

**Explanation**: - plotly.express.scatter\_geo creates an interactive map of airports, colored by country. - networkx models routes as a directed graph (DiGraph), where airports are nodes and routes are edges. - degree\_centrality measures the number of connections per airport, identifying key hubs. - A simple network visualization shows connections for a subset of routes (limited to 50 for clarity).

**Expected Outcome**: - An interactive map showing airport locations worldwide. - Network statistics (number of nodes and edges). - A list of the top 10 airports by connectivity (e.g., ATL, LAX). - A network graph visualizing connections between airports in the subset.

**Student Task**: - Save the interactive map as an HTML file (fig.write\_html('airport\_map.html')). - Identify the top airport by centrality and research why it’s a major hub (e.g., “ATL is a hub for Delta Airlines”). - Suggest one way to improve the network visualization (e.g., color nodes by country).

## Project Deliverables

To complete the course project, teams should submit:

1. **Code File**: A Python script or Jupyter Notebook containing all code from Tasks 1–5.

2. **Visualizations**: Saved charts (PNG for matplotlib, HTML for plotly) and the network graph.

3. **Report**: A 2–3 page document summarizing:

- Dataset description and cleaning steps.

- Key findings from EDA (e.g., top countries, airports).

- Insights from merged data (e.g., top airlines by routes).

- Network analysis results (e.g., top hub airports).

- Challenges faced and solutions applied.

1. **Technical Learning Reflections**: A 1-2 page document answering the technical learning questions posed on 22.
2. **Presentation**: A 5-minute slide deck summarizing the project for a class presentation.

**Team Collaboration Guidelines**

**Team Structure and Responsibilities**

**Team Members 1 and 2: Data Engineer**

* Lead data cleaning and preprocessing
* Manage data integration and relationship mapping
* Implement network analysis algorithms
* Ensure data quality and validation
* Create data pipelines and ETL processes

**Team Member 3: Data Analyst**

* Conduct exploratory data analysis
* Perform statistical analysis and hypothesis testing
* Lead business intelligence and market analysis
* Generate insights and recommendations
* Create analytical reports and summaries

**Team Member 4: Visualization Specialist**

* Design and implement all visualizations
* Create interactive dashboards and maps
* Lead geographic and spatial analysis
* Develop presentation materials
* Ensure visual consistency and user experience

**Communication and Coordination**

**Daily Standups** (15 minutes):

* What did you work on yesterday?
* What will you work on today?
* Any blockers or dependencies?

**Weekly Reviews** (1 hour):

* Demo progress and completed work
* Review and provide feedback
* Plan next week's priorities
* Address any issues or concerns

**Documentation Standards**:

* All code must be well-commented
* Use consistent naming conventions
* Document all data transformations and decisions

**Reflection Questions and Answers**

**Technical Learning Reflections**

Q1: What were the most significant data quality issues you encountered, and how did you address them?

Q2: Which analytical techniques provided the most valuable insights, and why?

Q3: How did you balance computational efficiency with analytical depth in your network analysis?

Collaboration and Process Reflections

Q4: How did your team handle conflicting approaches to analysis or interpretation?

Q5: What project management strategies were most effective for coordinating work across different specializations?

Q6: How did you ensure reproducibility and documentation quality throughout the project?

Industry and Business Insights Reflections

Q7: What surprised you most about the structure of the global aviation network?

Q8: How do your findings challenge or confirm common assumptions about airline competition?

Q9: What actionable recommendations would you make to airline executives based on your analysis?

Methodological and Analytical Reflections

Q10: How did you validate the accuracy and reliability of your network analysis results?

Q11: What limitations in your data or methodology should readers consider when interpreting your results?

Q12: How would you extend this analysis with additional data sources or more time?

# ****CanvasLMS-Lite: A Scalable, Python-Based Learning Management System****

## ****Project Objective****

CanvasLMS-Lite is a lightweight, scalable Learning Management System (LMS) built using Python. It is designed to replicate core features of Canvas, supporting role-based access for faculty and students. Key functionalities include assignment creation, submission, grading, simulated notifications, and messaging. This project emphasizes usability, security, and extensibility while leveraging standard Python libraries and simple file-based (JSON and text) data management.The system may also incorporates innovative features to enhance the learning experience, such as gamification and analytics dashboards.

Rather than implementing real-time email or messaging with external services, all communications (e.g., notifications and messages) are **simulated using functions** and **persisted in text or JSON files**. This design choice aligns with the educational purpose of the project and ensures it remains beginner-friendly.

### **Requirements and Specifications**

#### 1. User Authentication

The system implements secure user authentication with role-based registration and login.

* **ID Format**:
  + **Faculty**: 8-digit numeric IDs ending in "0000" (e.g., 12350000).
  + **Student**: 8-digit numeric IDs ending in a valid year between 2022 and 2028 (e.g., 12342024).
* **Password Requirements**:
  + Minimum 8 characters.
  + Must include at least one uppercase letter, one lowercase letter, one digit, and one special character (e.g., !@#$%^&\*).
  + Passwords are hashed (using bcrypt) before storage in users.json.
* **Registration**: Users provide their ID, full name, email address, and password during first login or registration. Input validation ensures compliance with ID and password rules.

#### 2. User Roles

The system supports two distinct roles with tailored functionalities and interfaces.

* **Faculty**:
  + Interface defaults to a dark theme for consistency and reduced eye strain.
  + Create assignments with title, description, due date, and optional file attachments.
  + View all student submissions in a dashboard with filters for assignment and student.
  + Grade submissions with numerical scores (0–100) and optional text feedback.
  + Simulate an automated email sent to students upon grading, including score and feedback.
* **Student**:
  + Choose between a colorful or dark theme during login or via profile settings.
  + View available assignments, including details and due dates.
  + Submit assignments as .txt or .py files, with automatic renaming and storage.
  + Access an in-app inbox to view notifications and grading feedback.

#### 3. GUI Features

The GUI may be developed using either:

* A **web-based interface** using Flask, HTML/CSS/JavaScript.
* A **desktop GUI** using eg. Tkinter.

Key features include:

* **Role-Based Dashboards**:
  + Faculty dashboard: Displays assignment creation forms, submission lists, grading tools, and messaging interfaces.
  + Student dashboard: Shows assignment lists, submission forms, grades, and messaging options.
* **Theme Customization**: Students can toggle between colorful and dark themes; faculty use a default dark theme.
* **Input Validation**: Real-time validation for ID formats, file types, and form inputs using JavaScript (web) or Python (Tkinter).
* **Scalability**: Python classes (e.g., User, Assignment, Submission, Message) manage core entities, ensuring modular and maintainable code.

#### 4. File Handling

The system securely manages assignment submissions with the following rules:

* **Accepted Formats**: Only .txt and .py files are permitted for submissions.
* **File Naming**: Submissions are renamed as studentID\_assignmentID.extension (e.g., 12342024\_hw1.py).
* **Storage**: Files are organized in a submissions directory with subfolders for each course and student (e.g., submissions/courseID/studentID/).
* **Timestamps**: Each submission is timestamped using the datetime module for tracking.
* **Faculty Access**: Faculty can view submission contents in the GUI (e.g., via a code viewer for .py files) and download files for offline review.

### ***5. Simulated Communication (Messaging & Notifications)***

All messaging and email functionalities will be **simulated using plain text files**:

**Submission Acknowledgements**: On submitting an assignment, a .txt file is written under a notifications/ folder with details for both student and faculty.

**Grading Feedback**: Grading triggers the creation of a feedback file readable by the student.

**In-App Inbox**: A dedicated interface displays sent and received messages (e.g., submission confirmations, grading notifications) for each user, stored in messages.json.

#### 7. Creativity and Innovation

To enhance the learning experience, the following innovative features are added:

* **Analytics Dashboard**:
  + Faculty view visualizations (e.g., bar charts of average grades per assignment) using **Plotly** or **Matplotlib**.
  + Students access a progress tracker showing grades, submission trends, and LP earned.
* **Course Announcements**:
  + Faculty can post course-wide announcements visible on the student dashboard, with email notifications for urgency.
* **Mobile Responsiveness**:
  + The Flask-based web interface is optimized for mobile devices using responsive CSS frameworks (e.g., Bootstrap), ensuring accessibility on smartphones and tablets.

#### 8. Data Management

Persistent data is stored in JSON files for simplicity and portability:

* **users.json**: Stores user ID, hashed password, name, email, role, and theme preference.
* **assignments.json**: Stores assignment ID, title, description, due date, and course ID.
* **submissions.json**: Stores student ID, assignment ID, file path, timestamp, grade, and feedback.
* **messages.json**: Stores sender ID, recipient ID, timestamp, and message content.
* **courses.json**: Stores course ID, title, faculty ID, and enrolled student IDs.

#### 9. Validation Rules

To ensure system integrity:

* IDs must be exactly 8 digits, validated using **re** (regular expressions).
* Faculty IDs end in 0000; student IDs end in a year between 2022 and 2028.
* Only .txt and .py files are accepted for submissions, enforced via file extension checks.
* Passwords are validated for length and complexity during registration.

#### 10. Integrity

The project adheres to academic integrity by implementing a unique design, leveraging custom Python classes and innovative features like analytics. Code reuse is avoided, and all functionalities are developed from scratch or with standard libraries.

#### 

#### 11. Suggested Python Modules

* **Flask**: Backend for the web-based GUI.
* **tkinter**: Alternative for desktop-based GUI.
* **smtplib, email**: For email notifications.
* **os, shutil**: For file and directory management.
* **datetime**: For timestamping submissions and messages.
* **re**: For ID and input validation.
* **json**: For persistent data storage.
* **bcrypt**: For password hashing.
* **pylint**: For automated code analysis (optional for .py submissions).
* **Plotly/Matplotlib**: For analytics visualizations.
* **Flask-SocketIO**: For real-time messaging (web-based).

### Implementation Plan

### Below is a high-level Python implementation outline for CanvasLMS-Lite, focusing on modularity and scalability using Flask or a GUI. The code uses Python classes and adheres to the specified requirements. Note that this is a simplified structure; a full implementation would require additional error handling, testing, and frontend assets (HTML/CSS/JavaScript).

Directory Structure

CanvasLMS-Lite/

├── core/ # Core logic and Python classes

│ ├── user.py

│ ├── assignment.py

│ ├── submission.py

│ ├── message.py

│ └── course.py

├── interface/ # GUI logic (Flask views or Tkinter forms)

├── data/ # JSON and text files for persistent storage

│ ├── users.json

│ ├── assignments.json

│ ├── submissions.json

│ ├── messages.json

│ └── courses.json

├── files/ # Submission uploads

├── utils/ # Helper functions (validation, file mang., etc.)

│ ├── notifier.py # Simulated notification system

│ ├── file\_handler.py

│ └── validator.py

├── templates/ (for Flask) # HTML templates

├── static/ (for Flask) # CSS, JS, images

└── main.py or app.py # Entry point

**Grading Rubric**

| **Category** | **Criteria** | **Points** | **Description of Expectations** |
| --- | --- | --- | --- |
| **1. Project Planning & Teamwork** | Roles & Task Allocation | 5 | Clear, fair distribution of tasks among team members; evidence of role definitions and responsibilities. |
|  | Collaboration & Communication | 5 | Demonstrated effective teamwork through documented meetings, version control commits, and integration of both project parts. |
| **2. Data Collection & Preprocessing (Aviation)** | Data Cleaning & Processing | 5 | Data cleaned thoroughly, missing values handled correctly, preprocessing steps clearly documented. |
| **3. Data Analysis & Insights (Aviation)** | Exploratory Data Analysis (EDA) | 10 | Comprehensive EDA using statistics and visualizations; trends and anomalies explored meaningfully. |
|  | Network Modeling & Interpretation | 10 | Network graphs constructed, key metrics calculated (e.g., centrality, connectivity); clear interpretation of results. |
| **4. Visualization & Reporting (Aviation)** | Visualizations | 5 | Visualizations are clear, well-labeled, meaningful, and enhance understanding of findings. |
|  | Insight Communication | 5 | Findings communicated in a professional report or notebook; key takeaways explained in plain language. |
| **5. Software Design (CanvasLMS-Lite)** | System Requirements & Architecture | 5 | Requirements clearly specified; system architecture diagram provided; design shows scalability. |
|  | Database Design | 5 | Logical database schema implemented (SQLite/MySQL); normalization principles applied. |
| **6. Implementation (CanvasLMS-Lite)** | GUI Design & Functionality | 10 | User interface is intuitive, user-friendly, and visually coherent; main features fully functional. |
|  | Core Features Implementation | 10 | Core LMS features implemented (course creation, user management, resource upload, tracking). |
|  | Code Quality & Documentation | 5 | Code is modular, readable, well-commented, with a clear README and usage instructions. |
| **7. Testing & Usability (CanvasLMS-Lite)** | System Testing | 5 | Evidence of testing for bugs, edge cases, and performance; test cases documented. |
|  | Usability & User Experience | 5 | Demonstrated attention to usability through feedback or usability checks; user experience aligns with design goals. |
| **8. Final Presentation & Deliverables** | Integrated Report & Submission | 5 | Complete project report, slides, code well-organized. |
|  | Final Demonstration | 5 | Live demonstration of the working LMS and EDA results; ability to answer questions clearly and confidently. |

| **TOTAL** | | **100** | |