AI TEAM TRAINING'26

TASK 2

Task 2.1: Decoding Driver Performance in Formula 1 History

The Scenario:

As part of the Formula 1 analytics division, you're working with a team of data scientists and engineers to gain a competitive edge through historical performance analysis. With over seven decades of racing data—from the grit and glory of Juan Manuel Fangio to the dominance of modern titans like Lewis Hamilton and Max Verstappen—there's a wealth of information waiting to be unlocked.

Your team is tasked with uncovering patterns, anomalies, and trends in driver and team performances from 1950 to 2024. The aim is to identify not just who performed best, but *why*—and what teams and drivers can learn from the past to shape the future.

Is there a certain era where DNFs were more common? Are some teams masters at consistency, finishing races with high efficiency, while others rely on risky, high-reward strategies? And which drivers consistently lost positions, despite promising qualifying runs?

This analysis will form the backbone of a performance intelligence report for technical directors, strategists, and media analysts alike.

Your Objective:

Use data from the historical race results (results.csv) to perform a comprehensive analysis that answers the following questions:

- Top Performers: Who are the top 10 drivers by total points?
- Team Efficiency: Which team scores the most points per race start?
- Reliability Analysis: Which season had the most DNFs, and what were the common causes?
- Worst Performers: Who lost the most positions from start to finish?
- Era Comparison: How do driver performances vary across different decades?

Technical Requirements:

- Data Cleaning and Preparation
- Feature Engineering (e.g., positions lost, race finish rate)
- Descriptive Statistics for Drivers and Races
- Outlier Detection and Handling

- Visualizations (bar charts, trend lines, heatmaps, etc.)
- Present findings in a clear, compelling report with insights and recommendations

---

Task 2.2: Uncovering the DNA of Formula 1 Car Performance with PCA

The Scenario:

You are a performance data analyst working with a Formula 1 engineering team preparing for next season's car design. The F1 world is fiercely competitive, where milliseconds matter, and each constructor brings a unique blend of design philosophy, engine power, aerodynamic efficiency, and budget constraints.

With countless technical specifications—horsepower, engine displacement, weight, downforce levels, acceleration, top speed, fuel efficiency under race conditions, and R&D cost—understanding what truly separates the top performers from the midfield and backmarkers is no trivial task.

Raw data alone is overwhelming and difficult to interpret. Is overall performance mainly a balance between raw speed and weight? Or are there hidden trade-offs—like high-downforce cornering specialists versus straight-line powerhouses?

Your team must decode the complex landscape of F1 car design to identify key performance archetypes. Doing so will guide strategic decisions about engineering focus and uncover competitive gaps your team can exploit.

Your Objective:

Your goal is to implement the Principal Component Analysis (PCA) algorithm **from scratch** to distill the high-dimensional dataset of Formula 1 car specifications into a few, uncorrelated **Principal Components**.

These components will represent the **core performance archetypes** of modern F1 cars such as "Aero-optimized Racer," "Power-focused Machine," or "Balanced All-Rounder." By reducing dimensionality while retaining the essence of the data, your analysis will help the engineering team:

- Understand the dominant design philosophies in the field.
- Visualize how different constructors compare.
- Strategically prioritize design trade-offs for next season's car.

Technical Requirements:

1. **Data Source:**
    - You must use the **"Cars 2022 Dataset"** from Kaggle:

- Use a relevant subset of the numerical columns for your analysis. Recommended features include: Horsepower, Torque, 0-60 MPH Time, MSRP, Fuel Economy, and Range.
- The Class column should be kept separately and used for color-coding the final visualization to see if the PCA-derived clusters align with human-defined car categories.

2. **Implementation (NumPy Version):**
    - You are forbidden from using pre-built PCA libraries like sklearn.decomposition.PCA.
    - You **must** use the NumPy library for numerical computations. Using numpy.linalg.eig for the eigen-decomposition step is permitted.
    - Your implementation must follow these distinct steps:
        i. **Data Loading and Preprocessing:** Load the dataset from Kaggle. Select only the relevant numerical feature columns for analysis (e.g., Horsepower, Torque, 0-60 MPH Time, Fuel Economy, Range, MSRP). Ensure any missing values are handled appropriately.
        ii. **Data Standardization:** For each feature, calculate the Z-score ($z = (x - \mu) / \sigma$) to scale the data.
        iii. **Covariance Matrix Calculation:** Compute the covariance matrix from the standardized data.
        iv. **Eigen-Decomposition:** Calculate the eigenvalues and eigenvectors of the covariance matrix.
        v. **Component Selection:** Sort the eigenvalues in descending order and sort their corresponding eigenvectors accordingly.
        vi. **Data Projection:** Construct a projection matrix from your selected eigenvectors. Multiply the original standardized data by this matrix to get the final, low-dimensional representation of the cars.
3. **Reporting and Visualization:**
    - Create a 2D scatter plot of the cars using their scores on the first two principal components. Each point should be color-coded by its Class (e.g., SUV, Sedan, Sports Car).
    - Write a comprehensive report in which you explain the concept of eigen values and eigen vectors, their application in the PCA. Also, explain the concept behind PCA. **Note for this part it is heavily discouraged to use ai-tools,** it is better if you do your own research and add resources for your summarized explanation.
    - Add your visualization results to the report.

Bonus Objective:

To gain a deep appreciation for the mechanics of PCA and the efficiency of numerical libraries, you will re-implement the key computational steps of the algorithm using only standard Python (lists of lists), without NumPy's matrix operations. You will then compare the performance of this manual implementation against your NumPy version, in terms of speed.

Technical Requirements for the Bonus:

1. **Pure Python Implementation:**
    - For this task, represent your data matrices as Python lists of lists.
    - You must write your own functions from scratch to perform the following matrix operations:
        - **Transpose a Matrix:** A function that takes a matrix and returns its transpose.
        - **Multiply two Matrices:** A function that performs the dot product of two matrices.
        - **Standardize Data:** A function that iterates through the columns of a matrix, calculates the mean and standard deviation for each, and returns the Z-score normalized matrix.
        - **Calculate the Covariance Matrix:** A function that takes the standardized data matrix and computes the covariance matrix using your manual matrix functions.
        - **Project Data:** Use your manual matrix multiplication function to project the data onto the chosen eigenvectors.
2. **Constraint:**
    - You are **still permitted to use libraries** to calculate the eigen values and vectors.
3. **Performance Benchmarking:**
    - Measure the time it takes to execute the entire PCA pipeline (from standardization to final projection) for both your NumPy version and your pure Python version, and also compare the output of both for validation and representing numerical errors.
    - In a dedicated section of your report, present the timing results in a table and explain *why* the NumPy version is drastically faster.

Task 2.3: Finding Hassan's Perfect Fit — Matching F1 Drivers to Team Philosophy

The Scenario

In the elite world of Formula 1, raw talent is only half the story. Teams aren't just looking for speed—they're searching for synergy. A driver must fit the team's culture, engineering philosophy, and racing DNA.

Meet Hassan, an emerging F1 talent known for his explosive pace, sharp adaptability, and strong team spirit. His performance traits have been quantified into a driver profile vector, capturing aspects like Speed, Aggression, Adaptability, Technical Skill, Teamwork, Risk-taking, and Consistency.

Three of the top teams—Red Bull, Ferrari, and Mercedes—each have their own unique identity, also expressed as feature vectors based on years of racing philosophy and driver strategy. Your role, as a data-driven performance analyst, is to determine which team Hassan aligns with most closely—not just statistically, but culturally.

Your Objective

Use vector similarity metrics to evaluate the compatibility between Hassan's driver profile and each team's identity:

- Hassan: [9, 8, 7, 6, 7, 8, 6]
- Red Bull: [10, 9, 6, 7, 6, 9, 5]
- Ferrari: [9, 7, 6, 6, 7, 7, 5]
- Mercedes: [8, 6, 8, 9, 9, 5, 9]

Technical Requirements

- Implement at least three different similarity metrics (e.g., Cosine Similarity, Euclidean Distance, Manhattan Distance, Pearson Correlation, etc.)
- Compute Hassan's similarity to each team.
- Recommend the team with the highest compatibility score.
- Write comments about each of these methods that includes:
  - Brief explanation of the similarity metrics used
  - The result of each method
  - Final recommendation based on comparative results

Bonus Objective

Each feature in the vector represents a human quality or skill. Instead of just using numbers, convert each trait (e.g., *Speed*, *Adaptability*, *Teamwork*) into word embeddings.

Intuition

An **embedding** is a numerical vector that represents the **meaning** of a word based on its context in language. Words with similar meanings (like "teamwork" and "collaboration") have **similar embeddings**—they're close in vector space.

We use embeddings to convert human traits like “speed” or “discipline” into numbers, so we can **mathematically compare** how similar Hassan is to each F1 team using tools like **cosine similarity**.

Steps:

- The vectors:
  - Hassan: ["speed", "aggression", "adaptability", "technical skill", "teamwork", "risk-taking", "consistency"]
  - Redbull: ["speed", "aggression", "adaptability", "technical skill", "teamwork", "risk-taking", "inconsistency"]
  - Ferrari: ["passion", "emotion", "adaptability", "technical skill", "teamwork", "risk-taking", "inconsistency"]
  - Mercedes: ["precision", "discipline", "adaptability", "technical skill", "teamwork", "control", "consistency"]
- Use a **pre-trained embedding model** to convert each word or phrase into a high-dimensional vector. (check tools like, GloVe, Word2Vec)
- Use a semantic similarity metric (e.g., cosine similarity) to compute compatibility between Hassan and each team based on the *meaning* of the traits, not just the numbers.
- Compare the results with the numeric method and reflect on the difference.

Submission Guidelines

- You will submit one zip file including your proposed solution to each one of the tasks in its own environment (directory/python files).
- Name each file with its task number.
- For illustration questions, you could either write a md file, or any readable text format with the task number.
- You can submit your code multiple times, only the last submission will be taken into consideration.
- The Task’s deadline is **3/8 11:59 PM**.
- It’s highly encouraged to do the bonus tasks
- Including functions, comments, and making your code clean, well-documented and readable will give bonus points.
- We are here to help!!

It is highly discouraged to cheat, the goal of the process is learning, so keep an eye for that.