## **1) End-to-end overview**

* **Data access (big CSVs) from Google Drive** using **DuckDB** → query huge files without loading everything to RAM.
* **Cleaning & outliers** → resulted in **fp\_clean** (our master, cleaned table).
* **Fast features**: numeric + booleans + simple date features, no heavy one-hot on full data.
* **ML (regression on totalFare)**:  
  + **Linear Regression**, **Ridge Regression**, **RandomForestRegressor**, **HistGradientBoostingRegressor**.
* **DL (two phases)**:  
  + **Regression** (predict numeric totalFare with small ANN/CNN/LSTM/LRCN/BiLSTM).
  + **Classification** (convert to “high vs low fare” using **training median** threshold → metrics: AUC/ROC/CM).
  + The **best classification ANN** was saved as **.h5** for the app.
* **Forecasting**: daily average fare by date using **SARIMAX**.
* **Streamlit app**: 4 tabs — **Prediction**, **Forecasting**, **Visualizations**, **Chatbot** (with demo input buttons).

## **2) Tools**

* **Google Colab** + **Google Drive**
* **DuckDB** (fast SQL over CSVs)
* **Python**: pandas, numpy, scikit-learn, tensorflow/keras, statsmodels, matplotlib
* **Streamlit** (simple UI)

**Why DuckDB?** It queries big CSVs directly, supports sampling (e.g., 500k rows), computes medians/quantiles, and can spill to disk to avoid OOM.

## **3) Data & features**

**Key columns**

* **Numeric**: baseFare, totalFare, totalTravelDistance, travelDuration, elapsedDays, seatsRemaining, segmentsDurationInSeconds, segmentsDistance
* **Booleans**: isBasicEconomy, isRefundable, isNonStop (cast to 0/1)
* **Dates**: searchDate, flightDate
* **High-cardinality strings** (airports/airlines/equipment/cabin): analyzed but **not** one-hot encoded at full scale.
* **ID**: legId (identifier only)

**Derived features**

* days\_ahead = flightDate − searchDate (days; fallback to elapsedDays if needed)
* flight\_is\_weekend (1 on Sat/Sun else 0)

These are the 12 inputs used in the app and many training runs:

baseFare,

totalTravelDistance,

travelDuration,

elapsedDays,

seatsRemaining,

segmentsDistance,

segmentsDurationInSeconds,

isNonStop\_bin,

isBasicEconomy\_bin,

isRefundable\_bin,

days\_ahead,

flight\_is\_weekend

## **4) Cleaning & outliers**

* **Missing values**
  + Numerics → **median** (DuckDB quantile(col, 0.5))
  + Booleans → **FALSE → 0**
  + Strings → left as-is for analysis; not used in fast modeling path
* **Outliers** on totalFare (IQR rule): keep rows within [Q1 − 1.5·IQR, Q3 + 1.5·IQR] → **fp\_clean**

We also enabled DuckDB disk spill (PRAGMA temp\_directory) when needed.

## **5) Train/test & sampling**

* Count eligible rows with non-null target, then sample **min(500,000, available)** for speed.
* **Train/validation split**: 80/20 (fixed random\_state for reproducibility).
* **NaN safety**: impute in DuckDB first; in pandas, coerce to numeric and fill any residuals; cast to float32 to keep RAM low.

## **6) Machine Learning (regression)**

Target: **continuous totalFare**. Models:

1. **Linear Regression** (baseline)
2. **Ridge Regression** (L2 regularization)
3. **RandomForestRegressor** (handles nonlinearity, robust)
4. **HistGradientBoostingRegressor** (fast gradient boosting on histograms)

**Metrics**: **MAE**, **RMSE**, **R²** on the validation split.  
 No scaling required for trees; linear models can benefit from scaling (optional).

## **7) Deep Learning (two modes)**

### **7.1 DL as regressors (on numeric totalFare)**

* Architectures (all minimal): **ANN**, **1D-CNN**, **LSTM**, **LRCN** (Conv1D→LSTM), **BiLSTM**
* Loss: **MSE**; report **MAE/RMSE**
* Inputs standardized with **StandardScaler**; early stopping for speed

### **7.2 DL as classifiers (high vs low fare)**

* Convert target to binary: **label = 1 if totalFare ≥ training median, else 0**
* Same architectures with **sigmoid** output and **binary\_crossentropy** loss
* **Metrics**: **AUC/ROC**, **Accuracy**, **Confusion Matrix**
* We compared ML & DL classifiers by AUC; **best model was the ANN classifier**, saved as **best\_model.h5**
  + In Streamlit we **load with compile=False** to avoid legacy metric deserialization issues.

So: **DL was used both as a regressor and a classifier** (two separate phases).

## **8) Forecasting (time series)**

* Build a daily series: **AVG(totalFare) by flightDate** (global; route filter optional).
* Fill calendar gaps by interpolation.
* **SARIMAX(1,1,1)(1,1,1,7)** with weekly seasonality.
* Validate on the last N days (e.g., 30): report **RMSE/MAPE**.
* Save the final forecaster to **/content/models/sarimax\_fare\_forecast.pkl**.

## **9) Streamlit app (4 tabs, demo buttons)**

**Paths (editable in sidebar):**

* ANN classifier: /content/models/best\_model.h5
* Scaler: /content/models/scaler.pkl
* Forecaster: /content/models/sarimax\_fare\_forecast.pkl

**Tabs**

1. **Prediction**
   * Loads **ANN classifier** + **scaler**
   * Single prediction form (the 12 features above)
   * **“Apply Demo Inputs 1”** button fills sensible values
   * Batch CSV upload → returns high\_fare\_prob
2. **Forecasting**
   * Loads **SARIMAX** forecaster
   * Choose horizon (days)
   * **“Apply Demo Forecast 1”** sets 30 days
   * Plots forecast and offers CSV download

**WHY and WHAT**

### **Linear Regression**

* **What it is: Think of it as drawing the best straight line through your data points. It assumes that the change in the target (like price) is proportional to the change in features (like distance, days before booking).**
* **Why we used it: It’s the simplest baseline model. It helps us understand whether there’s even a basic linear relationship in the dataset.**

### **2. Ridge Regression**

* **What it is: Same as linear regression but adds a penalty if the model puts too much weight on certain features. This prevents the model from “going crazy” with extreme values.**
* **Why we used it: In real datasets, many features overlap (multicollinearity). Ridge controls this and avoids overfitting, making the model more stable.**

### **3. Random Forest Regressor**

* **What it is: Instead of one big decision tree, it builds many small trees and averages their answers. Like asking 100 mechanics for advice and taking the average response.**
* **Why we used it: Real-world data isn’t perfectly linear. Random Forests capture complex interactions and usually perform well without heavy tuning.**

### **4. HistGradientBoosting Regressor**

* **What it is: A boosting method that builds trees one after another, each fixing the mistakes of the previous one. It also uses smart histogram binning for faster training on big data.**
* **Why we used it: Our dataset is large (500k+ rows). HistGradientBoosting is designed for speed and accuracy in such cases.**

# **🔹 Deep Learning Models**

### **1. Artificial Neural Network (ANN)**

* **What it is: A web of “neurons” (nodes) arranged in layers. Each layer processes the data and passes it to the next. Inspired by how our brain processes signals.**
* **Why we used it: ANNs can model complex, non-linear relationships beyond what ML models can easily handle. It’s our first deep learning baseline.**

### **2. Convolutional Neural Network (CNN)**

* **What it is: Uses filters that slide over data to detect local patterns (like edges in images, or repeating behaviors in sequences).**
* **Why we used it: Even though CNNs are famous for images, they also work well on structured/tabular data when we want to capture localized interactions between features.**

### **3. Long Short-Term Memory (LSTM)**

* **What it is: A type of Recurrent Neural Network (RNN) that remembers information for long sequences. It’s like having a memory of past events to predict the future.**
* **Why we used it: Our dataset has time aspects (searchDate, flightDate). LSTMs are perfect for time-dependent data because they can capture sequential patterns.**

### **4. LRCN (Long-term Recurrent Convolutional Network)**

* **What it is: Combines CNN + LSTM. CNN extracts features, LSTM handles sequence/time relationships.**
* **Why we used it: Sometimes we need both: CNN for detecting feature patterns and LSTM for tracking how they change over time.**

### **5. Bidirectional LSTM**

* **What it is: Reads sequences both forward and backward. Instead of only knowing the past, it also considers the future context in training.**
* **Why we used it: Flight/booking behavior can depend on both what came before (search date) and what comes after (flight date). BiLSTM captures both directions for richer context.**