🌐 Backend — FastAPI | FlightOnTime

Este projeto disponibiliza o modelo preditivo do **FlightOnTime** por meio de uma **API REST desenvolvida com FastAPI**, permitindo a previsão da probabilidade de atraso de um voo **antes da decolagem**, utilizando exclusivamente informações disponíveis no planejamento do voo.

O backend foi projetado como um **MVP técnico**, com foco em simplicidade, reprodutibilidade e clareza arquitetural, adequado para avaliação em hackathons e prototipagem rápida.

📁 Estrutura de Arquivos

A estrutura mínima do backend é organizada da seguinte forma:

```text
├── app.py
├── models/
│   └── model_flightontime.pkl
├── requirements.txt
└── README.md
```

O arquivo `app.py` contém toda a aplicação FastAPI. O diretório `models/` armazena o modelo treinado e serializado como um **pipeline completo** (pré-processamento + modelo). O arquivo `requirements.txt` lista todas as dependências necessárias para execução do serviço.

📦 Dependências do Projeto

As dependências utilizadas pelo backend são:

```
fastapi
uvicorn
pandas
numpy
scikit-learn
joblib
```

A instalação pode ser realizada com o comando:

```
pip install -r requirements.txt
```

🧠 Modelo Preditivo

O modelo utilizado é uma **Logistic Regression** treinada dentro de um **Pipeline do scikit-learn**, incluindo:

- Pré-processamento com `ColumnTransformer`
- One-Hot Encoding para variáveis categóricas
- Balanceamento de classes via `class_weight='balanced'`
- Ajuste de threshold para priorização de recall da classe atraso

O modelo é salvo no formato `.pkl` utilizando `joblib` e carregado diretamente pela API.

🚀 Implementação da API (app.py)

A aplicação FastAPI é implementada integralmente no arquivo `app.py`, conforme o código abaixo:

```
from fastapi import FastAPI
from pydantic import BaseModel
import pandas as pd
import joblib

app = FastAPI(
    title="FlightOnTime API",
    description="API para previsão de atraso de voos antes da
decolagem",
    version="1.0.0"
)

model = joblib.load("models/model_flightontime.pkl")

THRESHOLD = 0.4

class FlightInput(BaseModel):
```

```
    sigla_icao_empresa_aerea: str
    codigo_tipo_linha: str
    modelo_equipamento: str
    numero_de_assentos: int
    sigla_icao_aeroporto_origem: str
    sigla_icao_aeroporto_destino: str
    periodo_dia: str
    hora_partida_prevista: int

@app.post("/predict")
def predict_delay(flight: FlightInput):
    data = pd.DataFrame([flight.dict()])
    prob_atraso = model.predict_proba(data)[0][1]
    atraso_previsto = int(prob_atraso >= THRESHOLD)

    return {
        "atraso_previsto": atraso_previsto,
        "probabilidade_atraso": round(prob_atraso, 2),
        "threshold_utilizado": THRESHOLD
    }
```

▶️ Execução do Backend

Para iniciar o servidor localmente, execute o comando abaixo no diretório do projeto:

```
uvicorn app:app --reload
```

Após a execução, a API estará disponível em:

```
http://127.0.0.1:8000
```

🌐 Documentação Automática (Swagger)

O FastAPI gera automaticamente uma interface de documentação interativa. Para acessar o Swagger UI, utilize o endereço:

```
http://127.0.0.1:8000/docs
```

Uma documentação alternativa também pode ser acessada em:

`http://127.0.0.1:8000/redoc`

📥 Exemplo de Requisição

O endpoint `/predict` recebe um JSON com as informações pré-voo:

```
{
  "sigla_icao_empresa_aerea": "GLO",
  "codigo_tipo_linha": "N",
  "modelo_equipamento": "B738",
  "numero_de_assentos": 186,
  "sigla_icao_aeroporto_origem": "SBGR",
  "sigla_icao_aeroporto_destino": "SBRJ",
  "periodo_dia": "Tarde",
  "hora_partida_prevista": 14
}
```

📤 Exemplo de Resposta

A API retorna a classificação prevista, a probabilidade de atraso e o threshold utilizado:

```
{
  "atraso_previsto": 1,
  "probabilidade_atraso": 0.73,
  "threshold_utilizado": 0.4
}
```

🧪 Teste via Linha de Comando

A API pode ser testada diretamente via `curl`:

```
curl -X POST "http://127.0.0.1:8000/predict" \
-H "Content-Type: application/json" \
-d '{
  "sigla_icao_empresa_aerea": "GLO",
```

```
  "codigo_tipo_linha": "N",
  "modelo_equipamento": "B738",
  "numero_de_assentos": 186,
  "sigla_icao_aeroporto_origem": "SBGR",
  "sigla_icao_aeroporto_destino": "SBRJ",
  "periodo_dia": "Tarde",
  "hora_partida_prevista": 14
}'
```

🎚️ Estratégia de Threshold

O modelo retorna uma probabilidade de atraso. A classificação final é definida a partir de um **threshold fixado em 0.4**, escolhido para:

- Maximizar o recall da classe atraso
- Reduzir falsos negativos (atrasos não detectados)
- Atender ao objetivo operacional do problema