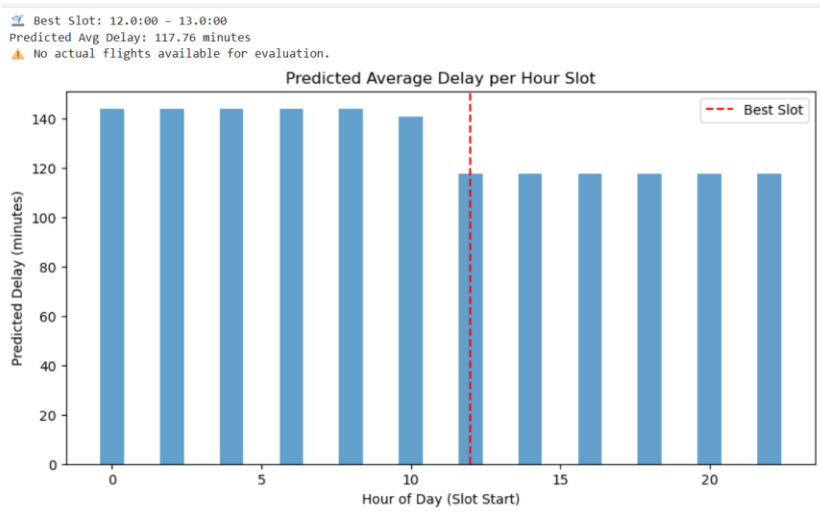
ML MODEL -RESULTS

METRICS:

	Model	MAE	RMSE	R2
0	RandomForest	20.201646	37.966809	0.581613
2	XGBoost	20.827460	37.075785	0.601020
1	GradientBoosting	21.050305	37.927764	0.582473
3	Ridge	31.976122	51.527597	0.229363
4	NeuralNet	32.015345	52.439434	0.201847

PREDICTIONS

```
Best Slot: 12.0:00 – 13.0:00
Predicted Avg Delay: 117.76 minutes
⚠ No actual flights available for evaluation.
```



EXPORT ML MODEL:

import os

os.makedirs("models", exist_ok=True) # create folder if not exists

joblib.dump(best_pipe, "models/best_pipe.pkl")

['models/best_pipe.pkl']

CODE:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import LabelEncoder

# === 0. Load / parse your dataframe (replace with your file) ===
# df = pd.read_csv("flights_clean.csv")  # or however you load it
# For safety, ensure columns exist and datetimes parse
df["STD"] = pd.to_datetime(df["STD"], errors="coerce")
df["ATD"] = pd.to_datetime(df["ATD"], errors="coerce")
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")

# Drop rows with essential missing values
df = df.dropna(subset=["STD", "ATD", "Date", "Aircraft", "From", "To"])

# === 1. Feature engineering ===
df["ScheduledHour"] = df["STD"].dt.hour
df["Weekday"] = df["Date"].dt.weekday

# Create route
df["Route"] = df["From"].astype(str) + "-" + df["To"].astype(str)

# Target: departure delay in minutes (float)
df["DepartureDelay"] = (df["ATD"] - df["STD"]).dt.total_seconds() / 60.0

# If you want to remove extreme outliers, consider clipping:
# df = df[(df["DepartureDelay"] > -60) & (df["DepartureDelay"] < 24*60)]

# === 2. Encode categorical features ===
```

```
le_aircraft = LabelEncoder()
le_route = LabelEncoder()

df["AircraftEncoded"] = le_aircraft.fit_transform(df["Aircraft"])
df["RouteEncoded"] = le_route.fit_transform(df["Route"])

# Save mappings for safe transform on unseen labels
aircraft_map = {lab: idx for idx, lab in enumerate(le_aircraft.classes_)}
route_map = {lab: idx for idx, lab in enumerate(le_route.classes_)}

# === 3. Prepare X, y and train/test split ===
feature_cols = ["ScheduledHour", "Weekday", "AircraftEncoded", "RouteEncoded"]
X = df[feature_cols].copy()
y = df["DepartureDelay"].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# === 4. Train model ===
model = RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1)
model.fit(X_train, y_train)

# Evaluate quickly
y_pred = model.predict(X_test)
print("MAE on test set: {:.2f} minutes".format(mean_absolute_error(y_test, y_pred)))

# === 5. Robust function to suggest best hour ===
def safe_encode_aircraft(a):
    # return encoded integer or -1 for unseen
    return aircraft_map.get(a, -1)

def safe_encode_route(from_code, to_code):
```

```
    route = f"{from_code}-{to_code}"
    return route_map.get(route, -1)

def suggest_best_time_for_flight(aircraft, origin, dest, weekday, candidate_hours=range(0,24)):
    """
    aircraft: string like "A320"
    origin, dest: strings e.g. "HYD", "DEL"
    weekday: int 0=Monday .. 6=Sunday (or use same convention as training)
    candidate_hours: iterable of hours to test (0..23)
    Returns (best_hour, predicted_delay_at_best_hour, df_with_all)
    """
    aircraft_enc = safe_encode_aircraft(aircraft)
    route_enc = safe_encode_route(origin, dest)
    if aircraft_enc == -1:
        print(f"Warning: aircraft '{aircraft}' not seen in training -> using -1 encoding.")
    if route_enc == -1:
        print(f"Warning: route '{origin}-{dest}' not seen in training -> using -1 encoding.")

    rows = []
    for h in candidate_hours:
        rows.append({
            "ScheduledHour": h,
            "Weekday": int(weekday),
            "AircraftEncoded": aircraft_enc,
            "RouteEncoded": route_enc
        })
    X_try = pd.DataFrame(rows, columns=feature_cols)
    preds = model.predict(X_try)
    X_try["PredictedDelay"] = preds
    best_idx = X_try["PredictedDelay"].idxmin()
    best_row = X_try.loc[best_idx]
```

```
    return int(best_row["ScheduledHour"]), float(best_row["PredictedDelay"]), X_try

# === 6. Example usage ===
# Example: a flight HYD -> DEL on Wednesday (weekday=2), aircraft "A320"
best_hour, best_pred_delay, candidates_df = suggest_best_time_for_flight(
    aircraft="A320",
    origin="HYD",
    dest="DEL",
    weekday=2
)

print("Best hour:", best_hour, "Predicted delay (min):", round(best_pred_delay, 2))
# candidates_df holds predicted delays for all 24 hours if you want to inspect/plot

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Models
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import Ridge
from sklearn.neural_network import MLPRegressor
import xgboost as xgb

# === Data preparation ===
df["STD"] = pd.to_datetime(df["STD"], errors="coerce")
df["ATD"] = pd.to_datetime(df["ATD"], errors="coerce")
```

```
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")

df = df.dropna(subset=["STD", "ATD", "Date", "Aircraft", "From", "To"])
df["ScheduledHour"] = df["STD"].dt.hour
df["Weekday"] = df["Date"].dt.weekday
df["Route"] = df["From"].astype(str) + "-" + df["To"].astype(str)
df["DepartureDelay"] = (df["ATD"] - df["STD"]).dt.total_seconds() / 60.0

# Features
cat_cols = ["Aircraft", "Route"]
num_cols = ["ScheduledHour", "Weekday"]
target = "DepartureDelay"

X = df[cat_cols + num_cols]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Preprocessing: OneHotEncoder for categorical vars
preprocessor = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
        ("num", "passthrough", num_cols)
    ]
)

# === Define candidate models ===
models = {
    "RandomForest": RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1),
```

```
    "GradientBoosting": GradientBoostingRegressor(n_estimators=300, random_state=42),
    "XGBoost": xgb.XGBRegressor(
        n_estimators=300, learning_rate=0.1, max_depth=6, random_state=42, n_jobs=-1
    ),
    "Ridge": Ridge(alpha=1.0),
    "NeuralNet": MLPRegressor(hidden_layer_sizes=(64,32), max_iter=500, random_state=42)
}

results = []

# === Train & Evaluate ===
for name, model in models.items():
    pipe = Pipeline(steps=[("pre", preprocessor), ("model", model)])
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))  # ✅ works in all versions
    r2 = r2_score(y_test, y_pred)

    results.append({
        "Model": name,
        "MAE": round(mae, 2),
        "RMSE": round(rmse, 2),
        "R2": round(r2, 3)
    })

results_df = pd.DataFrame(results).sort_values(by="MAE")
print(results_df)
#select best model
best_model_name = results_df.iloc[0]["Model"]
```

```
best_model = models[best_model_name]
best_pipe = Pipeline(steps=[("pre", preprocessor), ("model", best_model)])
best_pipe.fit(X, y)  # fit on full dataset

import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt

# ✅ Suggest best slot across a range
def suggest_best_time_slots(aircraft, route, weekday, slot_size, best_pipe, df):
    times = range(0, 24)  # scan full day
    predictions = []

    # 1. Predict delay for each hour
    for t in times:
        test_input = pd.DataFrame([{
            "Aircraft": aircraft,
            "Route": route,
            "Weekday": weekday,
            "ScheduledHour": t
        }])
        pred = best_pipe.predict(test_input)[0]
        predictions.append((t, pred))

    pred_df = pd.DataFrame(predictions, columns=["Hour", "PredictedDelay"])

    # 2. Group into slots (e.g. 6–7, 7–8 …)
    pred_df["SlotStart"] = (pred_df["Hour"] // slot_size) * slot_size
    slot_summary = pred_df.groupby("SlotStart")["PredictedDelay"].mean().reset_index()
```

```
    # 3. Find best slot (minimum avg delay)
    best_slot = slot_summary.loc[slot_summary["PredictedDelay"].idxmin()]

    # 4. Evaluate efficiency on actual data
    mask = (
        (df["Aircraft"] == aircraft) &
        (df["Route"] == route) &
        (df["Weekday"] == weekday)
    )
    actual_data = df.loc[mask, ["ScheduledHour", "DepartureDelay", "Aircraft", "Route", "Weekday"]]

    if not actual_data.empty:
        pred_delays = best_pipe.predict(actual_data[["Aircraft","Route","Weekday","ScheduledHour"]])
        actual_data = actual_data.assign(PredictedDelay=pred_delays)
        mae = mean_absolute_error(actual_data["DepartureDelay"], actual_data["PredictedDelay"])
        rmse = np.sqrt(mean_squared_error(actual_data["DepartureDelay"],
actual_data["PredictedDelay"]))
    else:
        mae, rmse = None, None

    return best_slot, slot_summary, actual_data, mae, rmse


# ✅ Example usage
best_slot, slot_summary, slot_df, mae, rmse = suggest_best_time_slots(
    df=df,
    aircraft="A320",
    route="MUM-DEL",
    weekday=0,      # Monday
    slot_size=2,    # 1-hour slots
    best_pipe=best_pipe
```

```
)

print(f"🛬 Best Slot: {best_slot['SlotStart']}:00 – {best_slot['SlotStart']+1}:00")
print(f"Predicted Avg Delay: {best_slot['PredictedDelay']:.2f} minutes")

if mae is not None:
    print(f"📊 Model Efficiency → MAE={mae:.2f}, RMSE={rmse:.2f}")
else:
    print("⚠️ No actual flights available for evaluation.")

# ✅ Plot all slots
plt.figure(figsize=(10,5))
plt.bar(slot_summary["SlotStart"], slot_summary["PredictedDelay"], width=0.8, alpha=0.7)
plt.axvline(best_slot["SlotStart"], color="red", linestyle="--", label="Best Slot")
plt.xlabel("Hour of Day (Slot Start)")
plt.ylabel("Predicted Delay (minutes)")
plt.title("Predicted Average Delay per Hour Slot")
plt.legend()
plt.show()
```