In [75]:
```
import pandas as pd
df=pd.read_csv('New_flightdata.csv',encoding="latin1")
df.to_csv("New_flightdata_utf8.csv", index=False, encoding="utf-8")

print("File saved as New_flightdata_utf8.csv with UTF-8 encoding")
df.head()
```

```
File saved as New_flightdata_utf8.csv with UTF-8 encoding
```

Out[75]:

	S.No	Flight Number	Date	From	To	Aircraft	Flight time	STD
0	1.0	AI2739	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	\n25 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQA)	1:34	9:00 AM
2	NaN	NaN	\n24 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNR)	1:36	9:00 AM
3	NaN	NaN	23-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQJ)	1:35	9:00 AM
4	NaN	NaN	22-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNF)	1:37	9:00 AM

In [71]:
```
df.columns
```

Out[71]:
```
Index(['S.No', 'Flight Number', 'Date', 'From', 'To', 'Aircraft',
       'Flight time', 'STD', 'ATD', 'STA', 'Unnamed: 10', 'ATA', 'Unnamed: 1
2',
       'Unnamed: 13'],
      dtype='object')
```

In [76]:
```
df = df.dropna(axis=1, how="all")

# Drop fully empty rows
df = df.dropna(axis=0, how="all")
```

In [74]:
```
df.head()
```

Out[74]:

	S.No	Flight Number	Date	From	To	Aircraft	Flight time	STD
0	1.0	AI2739	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	\n25 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQA)	1:34	9:00 AM
2	NaN	NaN	\n24 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNR)	1:36	9:00 AM
3	NaN	NaN	23-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQJ)	1:35	9:00 AM
4	NaN	NaN	22-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNF)	1:37	9:00 AM

In [77]:
```
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

# Check cleaned dataset
df.head()
```

Out[77]:

	S.No	Flight Number	Date	From	To	Aircraft	Flight time	STD
0	1.0	AI2739	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	\n25 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQA)	1:34	9:00 AM
2	NaN	NaN	\n24 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNR)	1:36	9:00 AM
3	NaN	NaN	23-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQJ)	1:35	9:00 AM
4	NaN	NaN	22-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNF)	1:37	9:00 AM

In [78]:
```
df = df.dropna(axis=1, how="all")

# Drop fully empty rows
df = df.dropna(axis=0, how="all")
```

In [79]:
```
df.head()
```

Out[79]:

	S.No	Flight Number	Date	From	To	Aircraft	Flight time	STD
0	1.0	AI2739	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	\n25 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQA)	1:34	9:00 AM
2	NaN	NaN	\n24 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNR)	1:36	9:00 AM
3	NaN	NaN	23-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQJ)	1:35	9:00 AM
4	NaN	NaN	22-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNF)	1:37	9:00 AM

In [80]:
```
df = df[['Flight Number','Date','From','To','Aircraft','Flight time','STD','AT
```

In [81]:
```
import pandas as pd

time_cols = ['STD', 'ATD', 'STA', 'ATA']

for col in time_cols:
    df[col] = (
        df[col]
        .astype(str)
        .str.replace("Landed", "", case=False)  # remove "Landed"
        .str.strip()
    )
    # Parse in 12-hour format
    df[col] = pd.to_datetime(df[col], format='%I:%M %p', errors='coerce')

def time_diff(t1, t2):
    if pd.isnull(t1) or pd.isnull(t2):
        return None
    return (t1 - t2).total_seconds() / 60  # difference in minutes

df['Dep_Delay'] = df.apply(lambda x: time_diff(x['ATD'], x['STD']), axis=1)
df['Arr_Delay'] = df.apply(lambda x: time_diff(x['ATA'], x['STA']), axis=1)
```

In [ ]:
```

```

In [82]:
```
df.head()
```

Out[82]:

	Flight Number	Date	From	To	Aircraft	Flight time	STD
0	AI2739	NaN	NaN	NaN	NaN	NaN	NaT
1	NaN	\n25 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQA)	1:34	1900-01-01 09:00:00
2	NaN	\n24 Jul 2025	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNR)	1:36	1900-01-01 09:00:00
3	NaN	23-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQJ)	1:35	1900-01-01 09:00:00
4	NaN	22-Jul-25	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNF)	1:37	1900-01-01 09:00:00

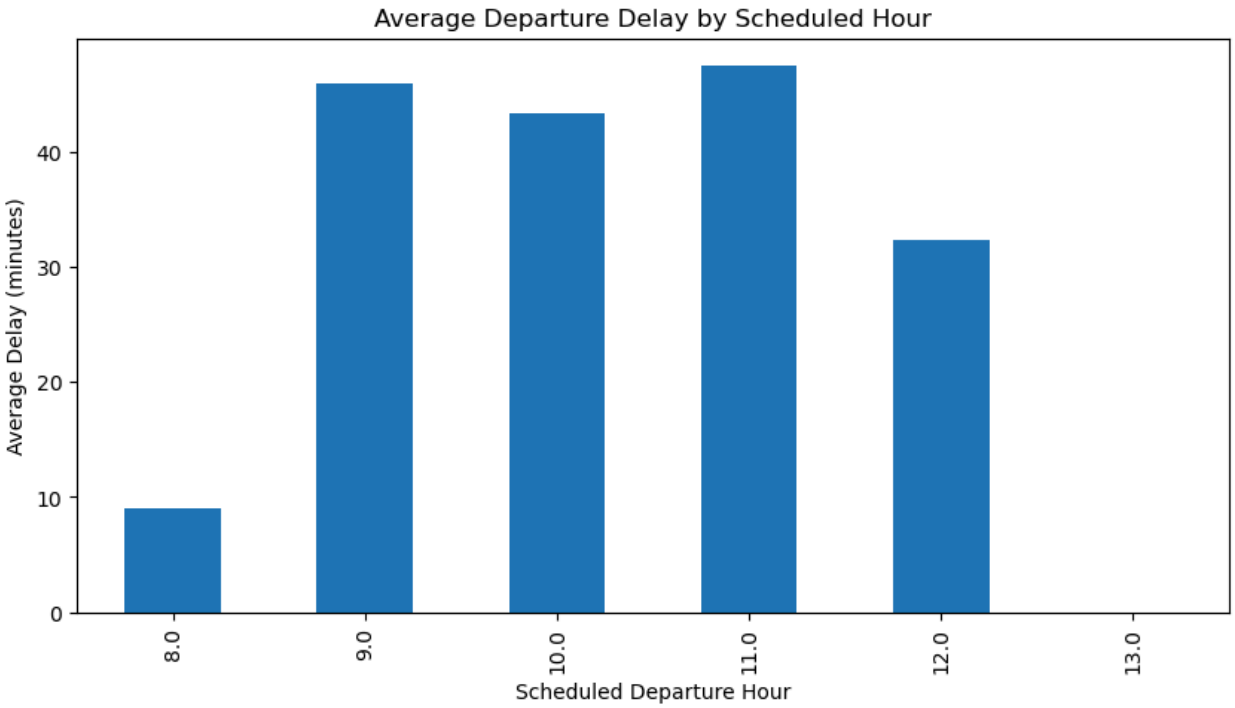
In [130…
```
df.columns
```

Out[130…
```
Index(['Flight Number', 'Date', 'From', 'To', 'Aircraft', 'Flight time', 'STD',
       'ATD', 'STA', 'ATA', 'Dep_Delay', 'Arr_Delay', 'STD_hour', 'weekday',
       'delay', 'takeoff_hour', 'landing_hour', 'takeoff_delay',
       'landing_delay', 'STA_hour', 'DayOfWeek', 'Route', 'ScheduledHour',
       'Weekday', 'AircraftEncoded', 'RouteEncoded', 'DepartureDelay'],
      dtype='object')
```

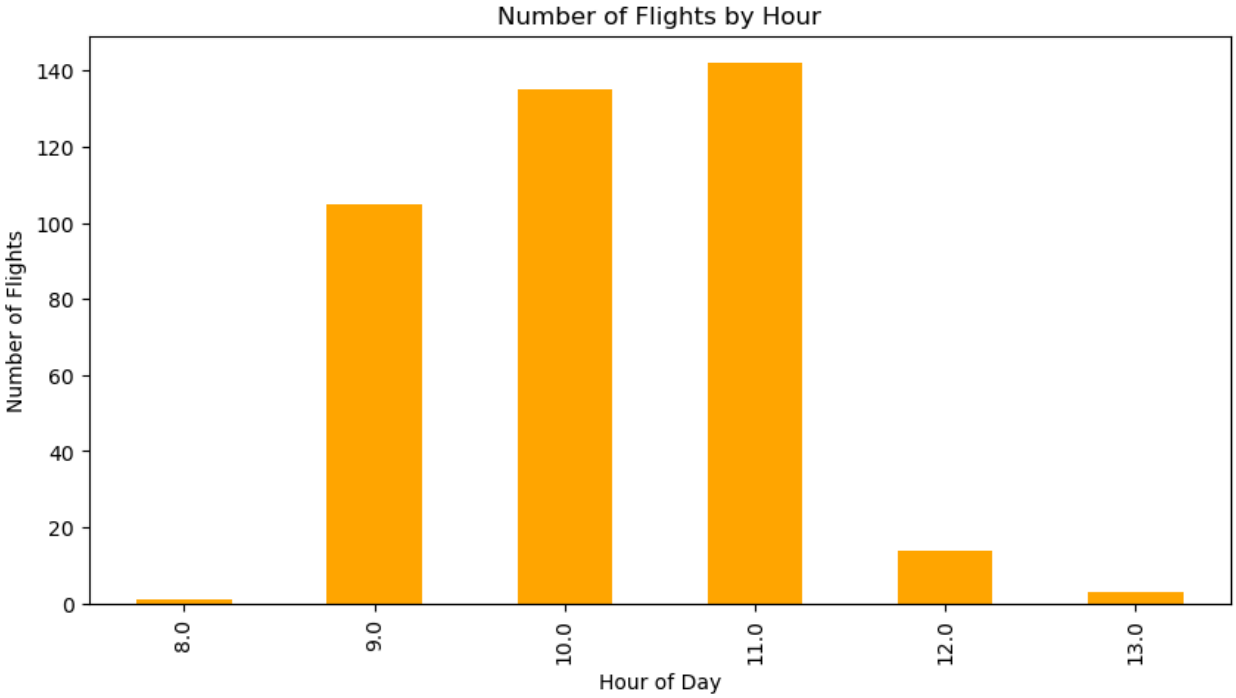
In [83]:
```
df.shape
```

Out[83]:
```
(458, 12)
```

In [85]:
```
#optimal hours
import matplotlib.pyplot as plt

df['STD_hour'] = pd.to_datetime(df['STD'], format='%H:%M:%S', errors='coerce').dt.hour
avg_delay_by_hour = df.groupby('STD_hour')['Dep_Delay'].mean()

plt.figure(figsize=(10,5))
avg_delay_by_hour.plot(kind='bar')
plt.title("Average Departure Delay by Scheduled Hour")
plt.xlabel("Scheduled Departure Hour")
plt.ylabel("Average Delay (minutes)")
plt.show()
```

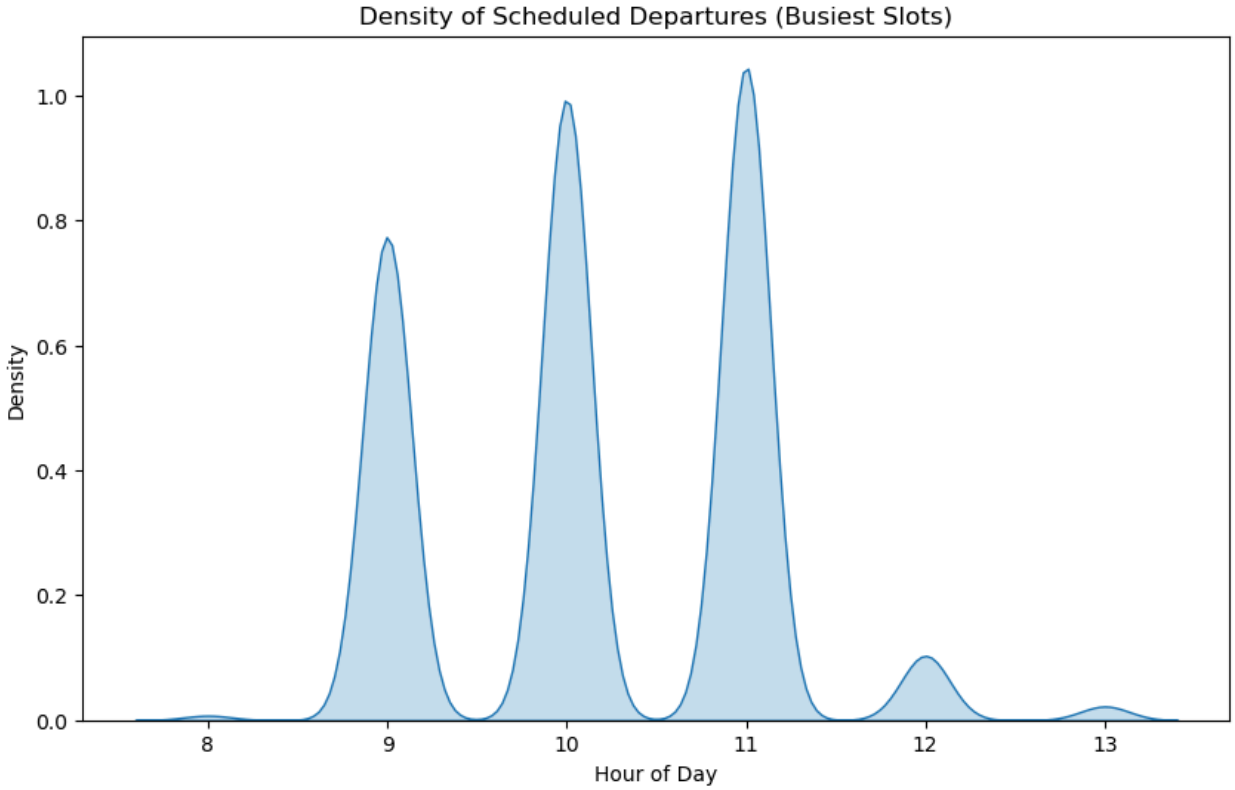


In [86]:
```
#busy hours
flight_counts = df['STD_hour'].value_counts().sort_index()

plt.figure(figsize=(10,5))
flight_counts.plot(kind='bar', color='orange')
plt.title("Number of Flights by Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Flights")
plt.show()
```



In [90]:
```
#Density plot for peak busy hour
plt.figure(figsize=(10,6))
sns.kdeplot(df['STD_hour'], shade=True, bw_adjust=0.5)
plt.title("Density of Scheduled Departures (Busiest Slots)")
plt.xlabel("Hour of Day")
plt.ylabel("Density")
plt.show()
```

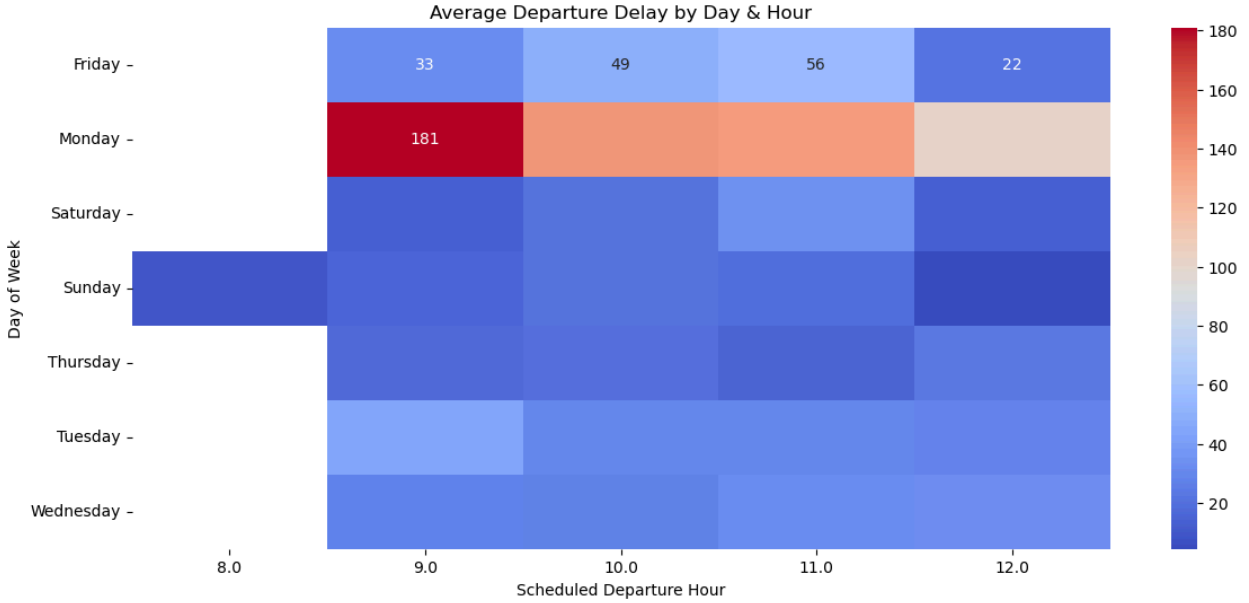
```
C:\Users\HP\AppData\Local\Temp\ipykernel_30660\1909742164.py:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['STD_hour'], shade=True, bw_adjust=0.5)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future versio
n. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [93]:
```
#Heat Map-for each day per hour delay rate
import seaborn as sns
import matplotlib.pyplot as plt

# Extract hour and weekday
df['STD_hour'] = pd.to_datetime(df['STD']).dt.hour
df['Date'] = pd.to_datetime(df['Date'])
df['weekday'] = df['Date'].dt.day_name()

# Calculate delay in minutes
df['delay'] = (pd.to_datetime(df['ATD']) - pd.to_datetime(df['STD'])).dt.total
```

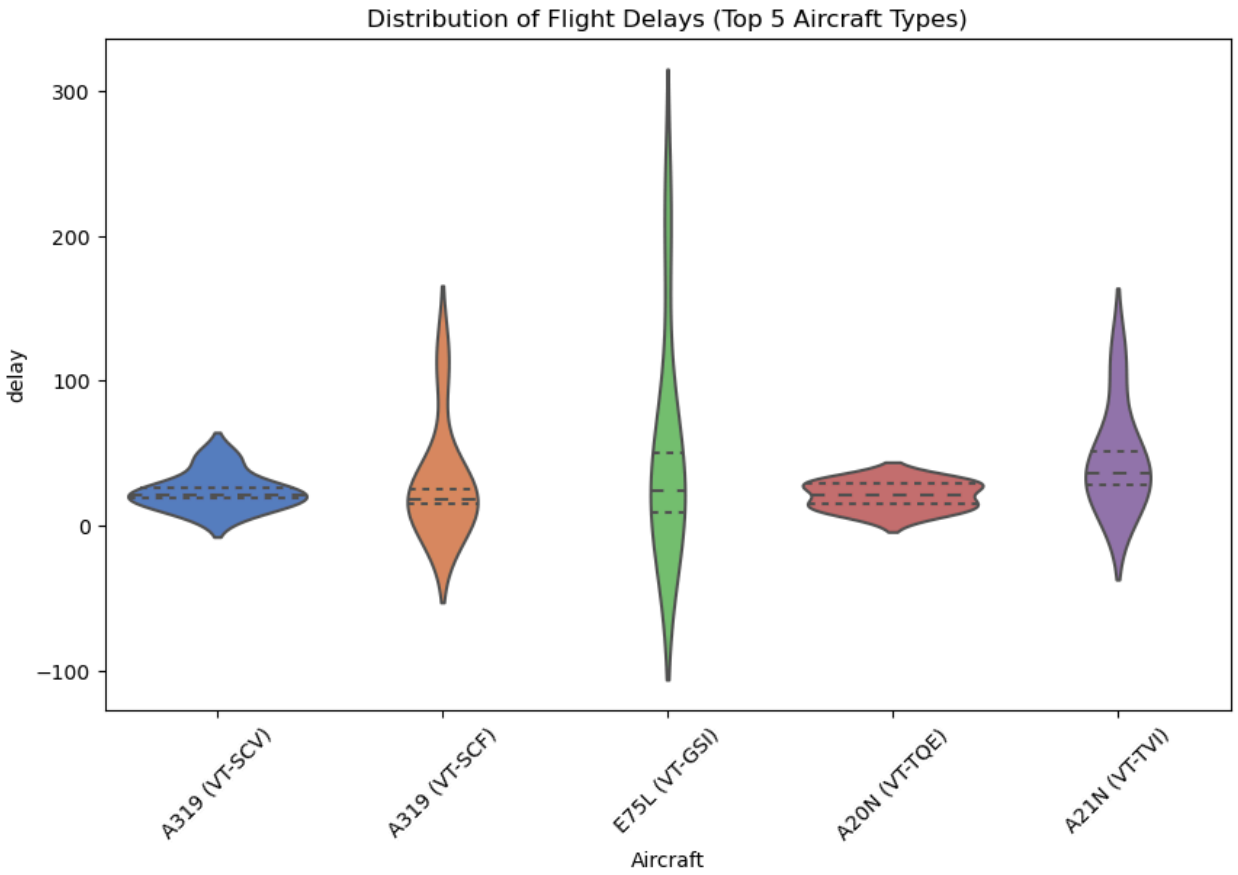
```
# Pivot for heatmap
heatmap_data = df.pivot_table(values='delay', index='weekday', columns='STD_ho

plt.figure(figsize=(14,6))
sns.heatmap(heatmap_data, cmap="coolwarm", annot=True, fmt=".0f")
plt.title("Average Departure Delay by Day & Hour")
plt.ylabel("Day of Week")
plt.xlabel("Scheduled Departure Hour")
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\matrix.py:260: FutureWarnin
g: Format strings passed to MaskedConstant are ignored, but in future may error
or produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```



In [92]:
```
df['Date'] = pd.to_datetime(df['Date'], format='%d-%b-%y', errors='coerce')
```

In [96]:
```
#Shows which aircraft type is more prone to delays.
import matplotlib.pyplot as plt
import seaborn as sns

# Find top 5 aircrafts by frequency
top_aircrafts = df["Aircraft"].value_counts().nlargest(5).index

# Filter dataset for top 5 aircrafts
df_top5 = df[df["Aircraft"].isin(top_aircrafts)]

# Plot violinplot for only top 5 aircrafts
plt.figure(figsize=(10,6))
sns.violinplot(data=df_top5, x="Aircraft", y="delay", inner="quartile", palett
plt.title("Distribution of Flight Delays (Top 5 Aircraft Types)")
plt.xticks(rotation=45)
plt.show()
```



In [97]:
```
df["takeoff_hour"] = pd.to_datetime(df["STD"]).dt.hour
df["landing_hour"] = pd.to_datetime(df["STA"]).dt.hour
```

In [98]:
```
df.head()
```

Out[98]:

	Flight Number	Date	From	To	Aircraft	Flight time	STD
0	AI2739	NaT	NaN	NaN	NaN	NaN	NaT
1	NaN	NaT	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQA)	1:34	1900-01-01 09:00:00
2	NaN	NaT	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNR)	1:36	1900-01-01 09:00:00
3	NaN	2025-07-23	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQJ)	1:35	1900-01-01 09:00:00
4	NaN	2025-07-22	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNF)	1:37	1900-01-01 09:00:00

In [99]:
```
takeoff_delay = df.groupby("takeoff_hour")["delay"].mean()
landing_delay = df.groupby("landing_hour")["delay"].mean()
```

In [100…
```
print(takeoff_delay)
```

```
takeoff_hour
8.0      9.000000
9.0     45.933333
10.0    43.343284
11.0    47.457143
12.0    32.285714
13.0          NaN
Name: delay, dtype: float64
```

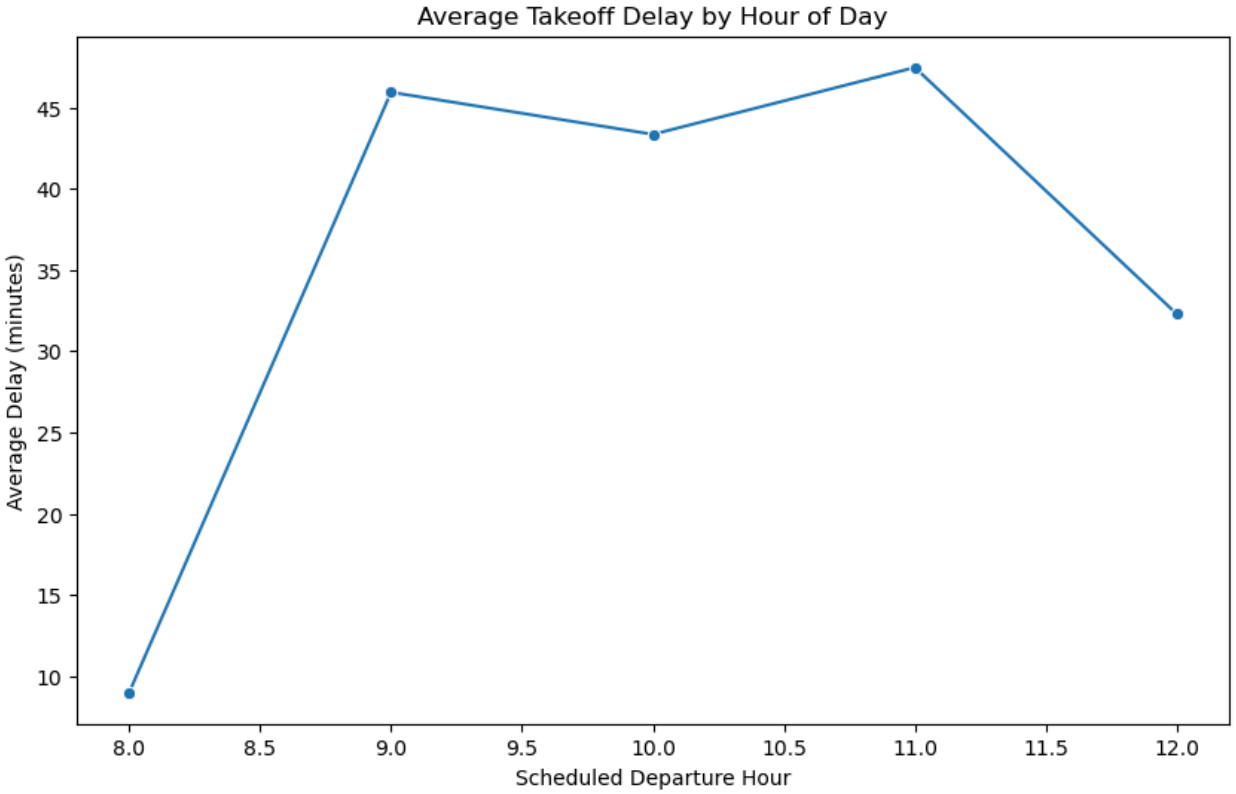
In [101…
```
print(landing_delay)
```

```
landing_hour
10.0    51.181818
11.0    43.263158
12.0    46.795455
13.0    43.965909
14.0    43.518519
16.0    40.571429
17.0    16.000000
18.0    52.000000
19.0    39.250000
Name: delay, dtype: float64
```

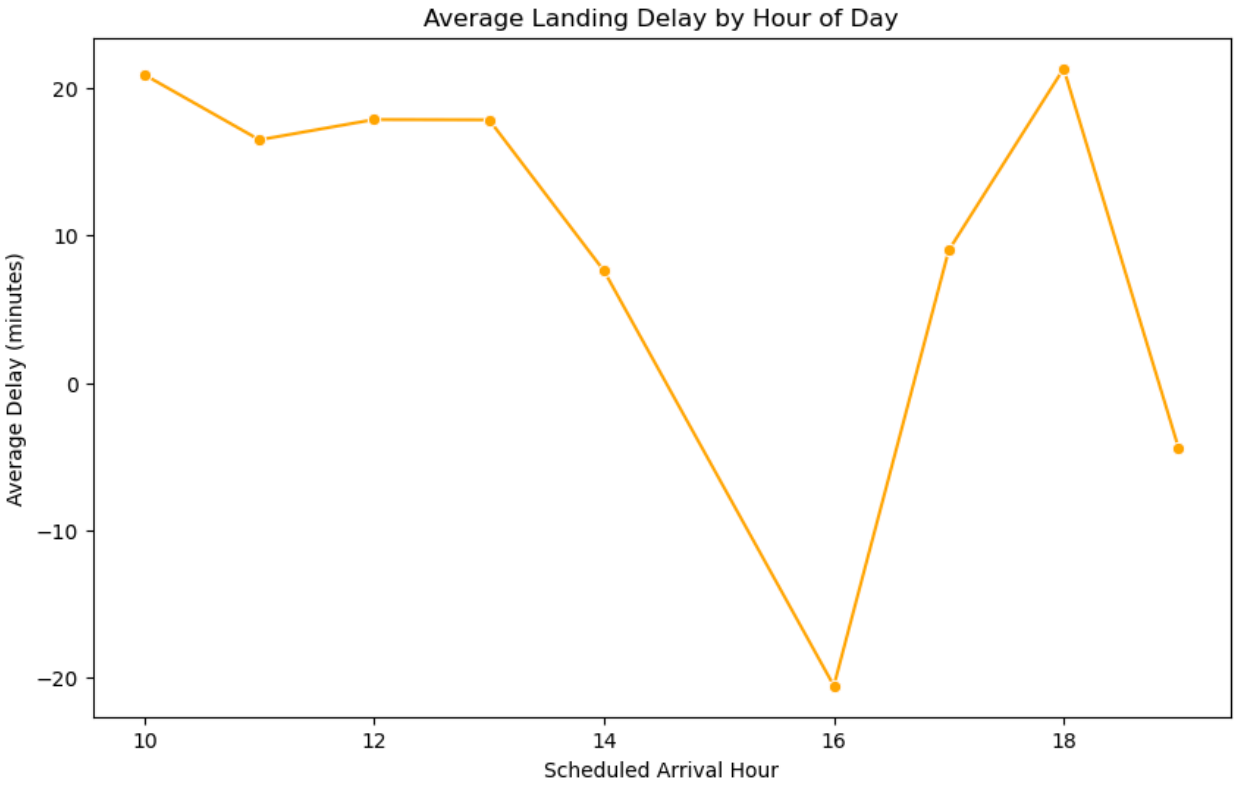
In [102…
```
##Best time for takeoff and landing
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Convert STD, ATD, STA, ATA to datetime if not already
df["STD"] = pd.to_datetime(df["STD"])
df["ATD"] = pd.to_datetime(df["ATD"])
df["STA"] = pd.to_datetime(df["STA"])
df["ATA"] = pd.to_datetime(df["ATA"])

# Calculate delays in minutes
df["takeoff_delay"] = (df["ATD"] - df["STD"]).dt.total_seconds() / 60
df["landing_delay"] = (df["ATA"] - df["STA"]).dt.total_seconds() / 60

# Extract scheduled hours
df["STD_hour"] = df["STD"].dt.hour
df["STA_hour"] = df["STA"].dt.hour

# Average delays per hour
takeoff_delay_by_hour = df.groupby("STD_hour")["takeoff_delay"].mean()
landing_delay_by_hour = df.groupby("STA_hour")["landing_delay"].mean()

# Plot best time for takeoff
plt.figure(figsize=(10,6))
sns.lineplot(x=takeoff_delay_by_hour.index, y=takeoff_delay_by_hour.values, ma
plt.title("Average Takeoff Delay by Hour of Day")
plt.xlabel("Scheduled Departure Hour")
plt.ylabel("Average Delay (minutes)")
plt.show()
```

```
# Plot best time for landing
plt.figure(figsize=(10,6))
sns.lineplot(x=landing_delay_by_hour.index, y=landing_delay_by_hour.values, ma
plt.title("Average Landing Delay by Hour of Day")
plt.xlabel("Scheduled Arrival Hour")
plt.ylabel("Average Delay (minutes)")
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future versio
n. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future versio
n. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

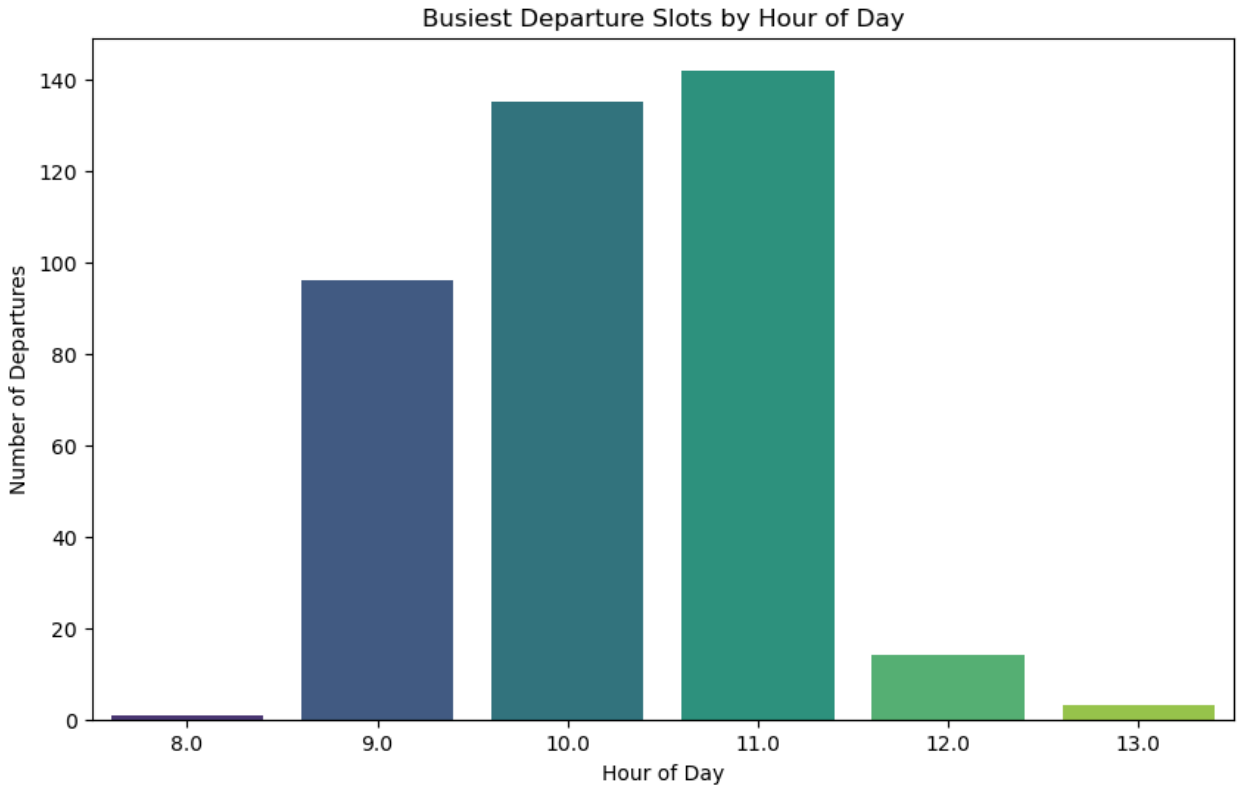



```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future versio
n. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future versio
n. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [103…

```
# Flights per hour (busiest slot)
flights_per_hour = df.groupby(df["STD"].dt.hour)["Flight Number"].count()

plt.figure(figsize=(10,6))
sns.barplot(x=flights_per_hour.index, y=flights_per_hour.values, palette="viri
plt.title("Busiest Departure Slots by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Departures")
plt.show()
```



In [111…

```python
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)]
```

```python
# === 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, randc

# === 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_p

# === 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_hc
    """
    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 -
    if route_enc == -1:
        print(f"Warning: route '{origin}-{dest}' not seen in training -> using

    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"]),

# === 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
# candidates_df holds predicted delays for all 24 hours if you want to inspect
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_30660\1969516336.py:11: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["STD"] = pd.to_datetime(df["STD"], errors="coerce")
C:\Users\HP\AppData\Local\Temp\ipykernel_30660\1969516336.py:12: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["ATD"] = pd.to_datetime(df["ATD"], errors="coerce")
C:\Users\HP\AppData\Local\Temp\ipykernel_30660\1969516336.py:13: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
MAE on test set: 20.50 minutes
Warning: aircraft 'A320' not seen in training -> using -1 encoding.
Warning: route 'HYD-DEL' not seen in training -> using -1 encoding.
Best hour: 11 Predicted delay (min): 20.32
```

In [113…

```
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, r
    "GradientBoosting": GradientBoostingRegressor(n_estimators=300, random_sta
    "XGBoost": xgb.XGBRegressor(
        n_estimators=300, learning_rate=0.1, max_depth=6, random_state=42, n_j
    ),
    "Ridge": Ridge(alpha=1.0),
```

```
    "NeuralNet": MLPRegressor(hidden_layer_sizes=(64,32), max_iter=500, random
}

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 vers
    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)
```

```
              Model    MAE   RMSE     R2
0      RandomForest  20.20  37.97  0.582
2           XGBoost  20.83  37.08  0.601
1  GradientBoosting  21.05  37.93  0.582
3             Ridge  31.98  51.53  0.229
4         NeuralNet  32.02  52.44  0.202
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_p
erceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(500) reached and the optimization hasn't converged yet.
  warnings.warn(
```

In [127…

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

X = df[cat_cols + num_cols].copy()
y = df[target].copy()

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

# Build pipelines for all models
models = {
```

```
    "RandomForest": RandomForestRegressor(n_estimators=200, random_state=42, r
    "GradientBoosting": GradientBoostingRegressor(n_estimators=300, random_sta
    "XGBoost": xgb.XGBRegressor(n_estimators=300, learning_rate=0.1, max_depth
                                random_state=42, n_jobs=-1),
    "Ridge": Ridge(alpha=1.0),
    "NeuralNet": MLPRegressor(hidden_layer_sizes=(64,32), max_iter=500, random
}

# Evaluate models
results = []
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))
    r2 = r2_score(y_test, y_pred)

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

results_df = pd.DataFrame(results).sort_values(by="MAE")
print(results_df)

# Select best
```

	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

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_p
erceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(500) reached and the optimization hasn't converged yet.
  warnings.warn(
```

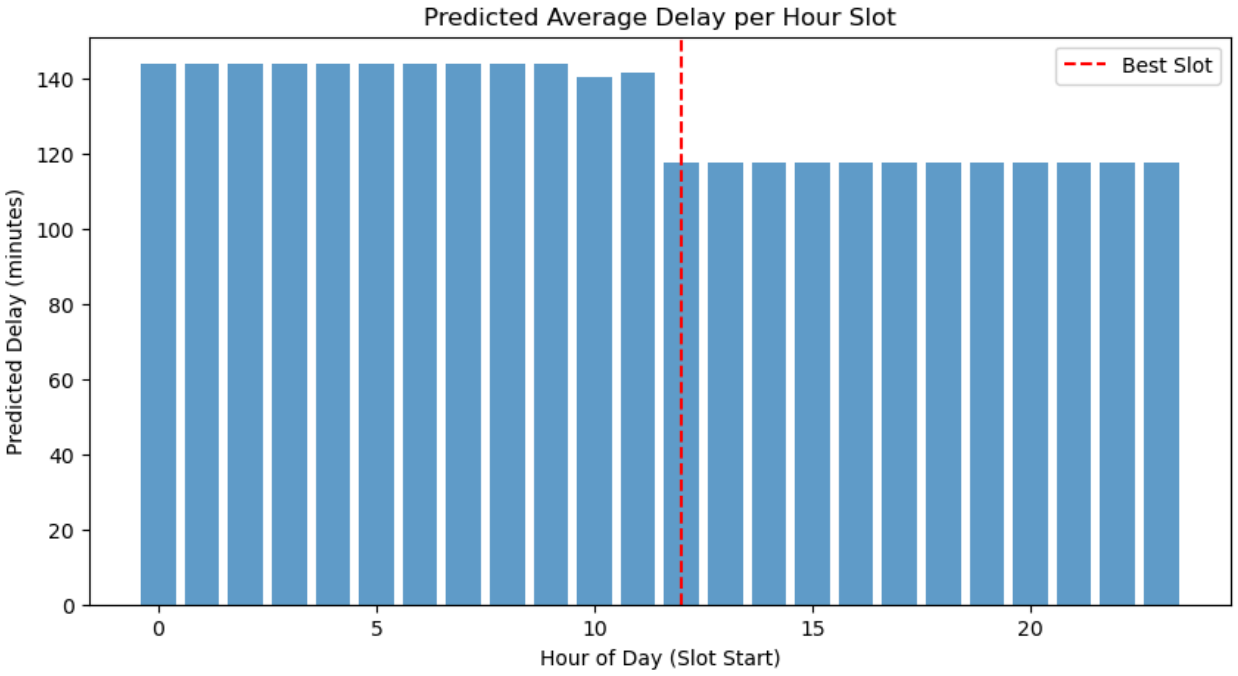
In [129…

```
#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

    # 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",

    if not actual_data.empty:
        pred_delays = best_pipe.predict(actual_data[["Aircraft","Route","Weekd
        actual_data = actual_data.assign(PredictedDelay=pred_delays)
        mae = mean_absolute_error(actual_data["DepartureDelay"], actual_data["
        rmse = np.sqrt(mean_squared_error(actual_data["DepartureDelay"], actua
    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=1,     # 1-hour slots
    best_pipe=best_pipe
)

print(f"✅ Best Slot: {best_slot['SlotStart']}:00 – {best_slot['SlotStart']+1}
```

```
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,
plt.axvline(best_slot["SlotStart"], color="red", linestyle="--", label="Best S
plt.xlabel("Hour of Day (Slot Start)")
plt.ylabel("Predicted Delay (minutes)")
plt.title("Predicted Average Delay per Hour Slot")
plt.legend()
plt.show()
```

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



In [ ]:

In [ ]:

In [ ]:

In [ ]:

model to predict flight that causes delays

In [131…

```
df.columns
```

Out[131…
```
Index(['Flight Number', 'Date', 'From', 'To', 'Aircraft', 'Flight time', 'STD',
       'ATD', 'STA', 'ATA', 'Dep_Delay', 'Arr_Delay', 'STD_hour', 'weekday',
       'delay', 'takeoff_hour', 'landing_hour', 'takeoff_delay',
       'landing_delay', 'STA_hour', 'DayOfWeek', 'Route', 'ScheduledHour',
       'Weekday', 'AircraftEncoded', 'RouteEncoded', 'DepartureDelay'],
      dtype='object')
```

In [132…
```
df.head()
```

Out[132…

	Flight Number	Date	From	To	Aircraft	Flight time	ST
3	NaN	2025-07-23	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TQJ)	1:35	1900-01-(09:00:(
4	NaN	2025-07-22	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNF)	1:37	1900-01-(09:00:(
6	NaN	2025-07-20	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNP)	1:33	1900-01-(09:00:(
7	NaN	2025-07-19	Mumbai (BOM)	Chennai (MAA)	A20N (VT-TNZ)	1:34	1900-01-(09:00:(
10	NaN	2025-07-25	Mumbai (BOM)	Cochin (COK)	B38M (VT-YAB)	1:36	1900-01-(09:10:(

5 rows × 27 columns

In [ ]:

In [ ]:

In [ ]:

In [135…
```
import pandas as pd
import networkx as nx

def build_flight_graph(df, turnaround=30):
    """
    Build dependency graph between flights based on aircraft rotations.
    """
    G = nx.DiGraph()

    # Ensure FlightID exists
    if "FlightID" not in df.columns:
        df = df.copy()
        df["FlightID"] = df.index.astype(str)

    # Add all flights as nodes
    for i, row in df.iterrows():
        G.add_node(row["FlightID"], **row.to_dict())
```

```
    # Connect flights by aircraft & time sequence
    for aircraft, group in df.groupby("Aircraft"):
        group = group.sort_values("STD")
        flights = group.to_dict("records")

        for i in range(len(flights) - 1):
            f1, f2 = flights[i], flights[i + 1]
            # If arrival + turnaround ≤ departure
            if pd.notnull(f1["ATA"]) and pd.notnull(f2["STD"]):
                if f1["ATA"] + pd.Timedelta(minutes=turnaround) <= f2["STD"]:
                    G.add_edge(f1["FlightID"], f2["FlightID"])
    return G


def compute_cascading_impact(G):
    """
    Compute direct + cascading delay impact for each flight.
    """
    impact_scores = {}

    for node in G.nodes:
        direct_delay = (G.nodes[node]["ATD"] - G.nodes[node]["STD"]).total_sec
        direct_delay = max(direct_delay, 0)

        # Propagate delays downstream
        indirect_delay = 0
        for downstream in nx.descendants(G, node):
            dep_delay = (G.nodes[downstream]["ATD"] - G.nodes[downstream]["STD
            indirect_delay += max(dep_delay, 0)

        impact_scores[node] = {
            "Aircraft": G.nodes[node]["Aircraft"],
            "DirectDelay": direct_delay,
            "IndirectDelay": indirect_delay,
            "ImpactScore": direct_delay + indirect_delay
        }

    return pd.DataFrame.from_dict(impact_scores, orient="index").sort_values("
```

In [136…

```
# Convert time columns
df["STD"] = pd.to_datetime(df["STD"])
df["ATD"] = pd.to_datetime(df["ATD"])
df["STA"] = pd.to_datetime(df["STA"])
df["ATA"] = pd.to_datetime(df["ATA"])

# Build network and compute cascading impact
G = build_flight_graph(df)
impact_df = compute_cascading_impact(G)

print("🔝 Top cascading impact flights:")
print(impact_df.head(10))
```

```
🔝 Top cascading impact flights:
          Aircraft  DirectDelay  IndirectDelay  ImpactScore
77   A20N (VT-IXH)        382.0              0        382.0
14   B38M (VT-YAF)        350.0              0        350.0
416  AT76 (VT-AIY)        295.0              0        295.0
114  A20N (VT-TYB)        271.0              0        271.0
68   E75L (VT-GSI)        213.0              0        213.0
254  A21N (VT-ICF)        206.0              0        206.0
58   B772 (G-YMMT)        205.0              0        205.0
245  A21N (VT-IML)        196.0              0        196.0
423  A20N (VT-TNU)        193.0              0        193.0
105  A20N (VT-IIU)        182.0              0        182.0
```

In [138…

```python
import os

os.makedirs("models", exist_ok=True)  # create folder if not exists
joblib.dump(best_pipe, "models/best_pipe.pkl")
```

Out[138…

```
['models/best_pipe.pkl']
```

In [ ]: