Project_code

December 2, 2025

[60]:
```
import numpy as np
import pandas as pd
import zipfile

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score,
    precision_recall_fscore_support,
    roc_auc_score,
    confusion_matrix
)
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer
```

[61]:
```
import matplotlib.pyplot as plt


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
```

[62]:
```
ZIP_PATH = "archive (16).zip"              # your uploaded zip file
CSV_NAME = "flight_data_2018_to_2022.csv"  # inside the zip

with zipfile.ZipFile(ZIP_PATH, "r") as z:
    with z.open(CSV_NAME) as f:
        df = pd.read_csv(f, low_memory=False)
```

```
print("Full shape:", df.shape)
df.head()
```

```
Full shape: (563737, 120)
```

[62]:

	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	\
0	2022	1	1	6	4	2022-01-06	
1	2022	1	1	6	4	2022-01-06	
2	2022	1	1	6	4	2022-01-06	
3	2022	1	1	6	4	2022-01-06	
4	2022	1	1	6	4	2022-01-06	

	Marketing_Airline_Network	Operated_or_Branded_Code_Share_Partners	\
0	DL	DL	
1	DL	DL	
2	DL	DL	
3	DL	DL	
4	DL	DL	

	DOT_ID_Marketing_Airline	IATA_Code_Marketing_Airline	…	Div5Airport	\
0	19790	DL	…	NaN	
1	19790	DL	…	NaN	
2	19790	DL	…	NaN	
3	19790	DL	…	NaN	
4	19790	DL	…	NaN	

	Div5AirportID	Div5AirportSeqID	Div5WheelsOn	Div5TotalGTime	\
0	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	

	Div5LongestGTime	Div5WheelsOff	Div5TailNum	Duplicate	Unnamed: 119
0	NaN	NaN	NaN	N	NaN
1	NaN	NaN	NaN	N	NaN
2	NaN	NaN	NaN	N	NaN
3	NaN	NaN	NaN	N	NaN
4	NaN	NaN	NaN	N	NaN

[5 rows x 120 columns]

[63]:
```
# Keep only rows where ArrDel15 is not missing
df = df[df["ArrDel15"].notna()].copy()

# Make sure ArrDel15 is int 0/1
df["ArrDel15"] = df["ArrDel15"].astype(int)
```

```
print("Shape after dropping NaN ArrDel15:", df.shape)
print("Delay rate (ArrDel15=1):", df["ArrDel15"].mean())
```

```
Shape after dropping NaN ArrDel15: (526894, 120)
Delay rate (ArrDel15=1): 0.19456475116437083
```

[64]:
```
def hhmm_to_minutes(x):
    """
    Convert HHMM numeric time (e.g., 1126) to minutes since midnight.
    Returns NaN if invalid.
    """
    try:
        x_int = int(x)
    except (ValueError, TypeError):
        return np.nan
    hour = x_int // 100
    minute = x_int % 100
    if hour < 0 or hour > 23 or minute < 0 or minute > 59:
        return np.nan
    return hour * 60 + minute

df["CRSDepMinutes"] = df["CRSDepTime"].apply(hhmm_to_minutes)

# Features (only pre-departure info)
feature_cols = [
    "Month",
    "DayOfWeek",
    "CRSDepMinutes",
    "Distance",
    "Origin",
    "Dest",
    "IATA_Code_Marketing_Airline"
]
target_col = "ArrDel15"

data = df[feature_cols + [target_col]].copy()
print("Selected columns:", data.columns.tolist())
data.head()
```

```
Selected columns: ['Month', 'DayOfWeek', 'CRSDepMinutes', 'Distance', 'Origin',
'Dest', 'IATA_Code_Marketing_Airline', 'ArrDel15']
```

[64]:

	Month	DayOfWeek	CRSDepMinutes	Distance	Origin	Dest	\
1	1	4	991	581.0	ATL	FLL	
2	1	4	1171	581.0	FLL	ATL	
3	1	4	624	680.0	FLL	RDU	

```
4      1          4            677     341.0    ATL  JAN
5      1          4            757     341.0    JAN  ATL

  IATA_Code_Marketing_Airline  ArrDel15
1                          DL         0
2                          DL         0
3                          DL         0
4                          DL         0
5                          DL         0
```

0.1 overall delay rate from data (EDA)

[65]:
```python
# EDA Cell 1: Overall delay rate in the sampled data
overall_delay_rate = data["ArrDel15"].mean()
print(f"Overall delay rate (ArrDel15=1): {overall_delay_rate:.3f}")
```

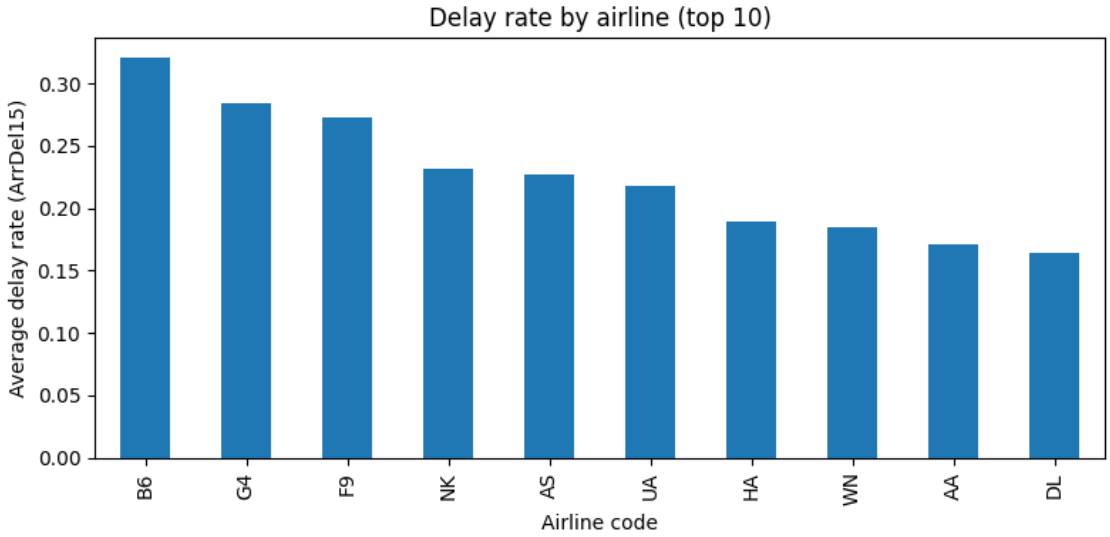
```
Overall delay rate (ArrDel15=1): 0.195
```

0.2 delay rate by airline + bar plot

[66]:
```python
# EDA Cell 2: Delay rate by airline (IATA_Code_Marketing_Airline)

delay_by_airline = (
    data.groupby("IATA_Code_Marketing_Airline")["ArrDel15"]
        .mean()
        .sort_values(ascending=False)
)

print("Top 10 airlines by delay rate:")
print(delay_by_airline.head(10))

# Plot 1: Bar chart for top 10 airlines by delay rate
plt.figure(figsize=(8, 4))
delay_by_airline.head(10).plot(kind="bar")
plt.xlabel("Airline code")
plt.ylabel("Average delay rate (ArrDel15)")
plt.title("Delay rate by airline (top 10)")
plt.tight_layout()
plt.show()
```

```
Top 10 airlines by delay rate:
IATA_Code_Marketing_Airline
B6    0.320185
G4    0.284138
F9    0.272767
```

```
NK    0.232032
AS    0.226563
UA    0.217331
HA    0.189539
WN    0.184186
AA    0.170686
DL    0.164472
Name: ArrDel15, dtype: float64
```

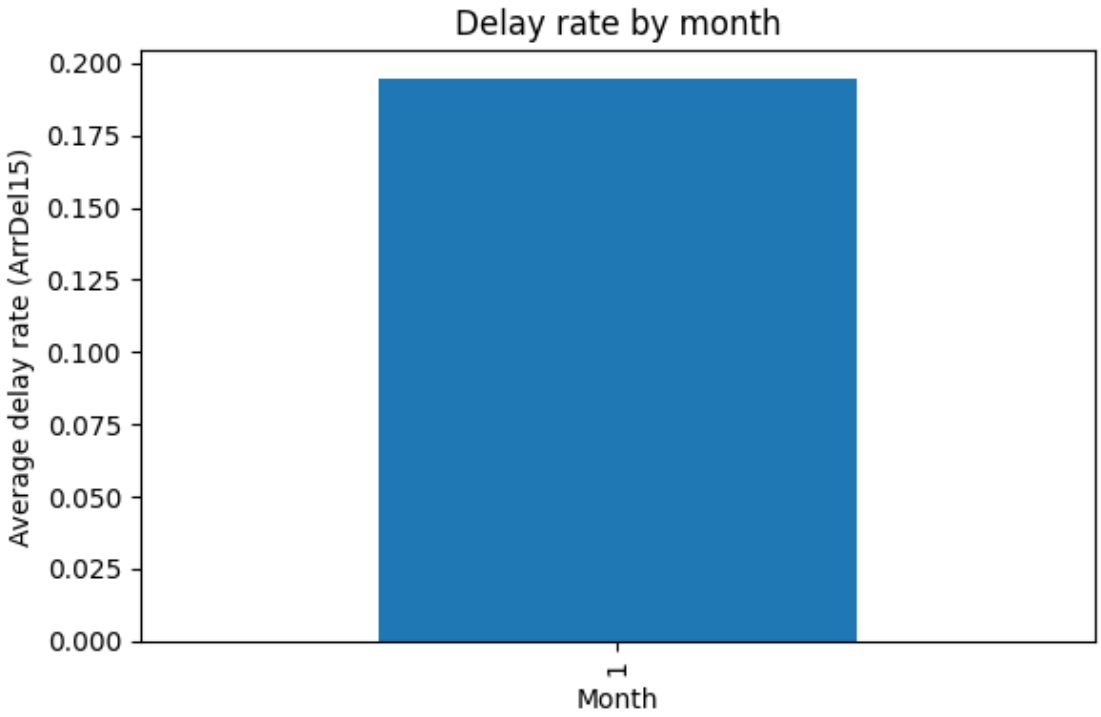


0.3 delay rate by month + bar plot

[67]:
```
# EDA Cell 3: Delay rate by month (seasonality)

delay_by_month = data.groupby("Month")["ArrDel15"].mean()
print("Delay rate by month:")
print(delay_by_month)

# Optional small plot for your understanding (you can screenshot if you like)
plt.figure(figsize=(6, 4))
delay_by_month.plot(kind="bar")
plt.xlabel("Month")
plt.ylabel("Average delay rate (ArrDel15)")
plt.title("Delay rate by month")
plt.tight_layout()
plt.show()
```

```
Delay rate by month:
Month
```

```
1    0.194565
Name: ArrDel15, dtype: float64
```



[68]:
```
USE_SAMPLE = True          # set False to use all data
N_SAMPLES = 150_000        # adjust based on RAM / speed

if USE_SAMPLE and len(data) > N_SAMPLES:
    data = data.sample(n=N_SAMPLES, random_state=42).copy()
    print("Using sample:", data.shape)
else:
    print("Using full dataset:", data.shape)
```

```
Using sample: (150000, 8)
```

0.4 define X, y and train_test_split (80/20, stratified)

[69]:
```
X = data[feature_cols]
y = data[target_col]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
```

```
    random_state=42
)

print("Train size:", X_train.shape, " Test size:", X_test.shape)
print("Train delay rate:", y_train.mean())
print("Test delay rate :", y_test.mean())
```

```
Train size: (120000, 7)  Test size: (30000, 7)
Train delay rate: 0.19303333333333333
Test delay rate : 0.19303333333333333
```

0.5 Preprocessing

[70]:
```
numeric_features = ["Month", "DayOfWeek", "CRSDepMinutes", "Distance"]
categorical_features = ["Origin", "Dest", "IATA_Code_Marketing_Airline"]

def make_preprocess():
    numeric_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
    ])

    categorical_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("onehot", OneHotEncoder(handle_unknown="ignore"))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", numeric_transformer, numeric_features),
            ("cat", categorical_transformer, categorical_features)
        ]
    )
    return preprocessor
```

[71]:
```
def evaluate_model(name, y_true, y_pred, y_proba=None):
    acc = accuracy_score(y_true, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(
        y_true, y_pred,
        average="binary",
        pos_label=1,
        zero_division=0
    )
    if y_proba is not None:
        try:
            auc = roc_auc_score(y_true, y_proba)
```

```
        except ValueError:
            auc = np.nan
    else:
        auc = np.nan

    print(f"\n=== {name} ===")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall   : {recall:.4f}")
    print(f"F1-score : {f1:.4f}")
    print(f"ROC-AUC  : {auc:.4f}")
    print("Confusion matrix:")
    print(confusion_matrix(y_true, y_pred))

    return {
        "model": name,
        "accuracy": acc,
        "precision": precision,
        "recall": recall,
        "f1": f1,
        "roc_auc": auc
    }

results = []
```

0.6 Baseline models

[72]:
```
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train, y_train)
y_pred_dummy = dummy_clf.predict(X_test)

results.append(evaluate_model("Majority Class (Dummy)", y_test, y_pred_dummy))
```

```
=== Majority Class (Dummy) ===
Accuracy : 0.8070
Precision: 0.0000
Recall   : 0.0000
F1-score : 0.0000
ROC-AUC  : nan
Confusion matrix:
[[24209     0]
 [ 5791     0]]
```

[73]:
```
# Logistic Regression
log_reg = Pipeline(steps=[
```

```
    ("preprocess", make_preprocess()),
    ("clf", LogisticRegression(
        max_iter=1000,
        class_weight="balanced",
        n_jobs=-1
    ))
])

log_reg.fit(X_train, y_train)
y_pred_lr = log_reg.predict(X_test)
y_proba_lr = log_reg.predict_proba(X_test)[:, 1]
results.append(evaluate_model("Logistic Regression", y_test, y_pred_lr,␣
 ↪y_proba_lr))

# Decision Tree
dt_clf = Pipeline(steps=[
    ("preprocess", make_preprocess()),
    ("clf", DecisionTreeClassifier(
        max_depth=10,
        class_weight="balanced",
        random_state=42
    ))
])

dt_clf.fit(X_train, y_train)
y_pred_dt = dt_clf.predict(X_test)
y_proba_dt = dt_clf.predict_proba(X_test)[:, 1]
results.append(evaluate_model("Decision Tree", y_test, y_pred_dt, y_proba_dt))
```

```
=== Logistic Regression ===
Accuracy : 0.5977
Precision: 0.2592
Recall   : 0.5833
F1-score : 0.3589
ROC-AUC  : 0.6277
Confusion matrix:
[[14553  9656]
 [ 2413  3378]]

=== Decision Tree ===
Accuracy : 0.5751
Precision: 0.2535
Recall   : 0.6177
F1-score : 0.3595
ROC-AUC  : 0.6232
Confusion matrix:
```

```
[[13676 10533]
 [ 2214  3577]]
```

[74]:
```python
# Random Forest
rf_clf = Pipeline(steps=[
    ("preprocess", make_preprocess()),
    ("clf", RandomForestClassifier(
        n_estimators=200,
        max_depth=15,
        class_weight="balanced_subsample",
        n_jobs=-1,
        random_state=42
    ))
])

rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)
y_proba_rf = rf_clf.predict_proba(X_test)[:, 1]
results.append(evaluate_model("Random Forest", y_test, y_pred_rf, y_proba_rf))
```

```
=== Random Forest ===
Accuracy : 0.6366
Precision: 0.2758
Recall   : 0.5431
F1-score : 0.3658
ROC-AUC  : 0.6437
Confusion matrix:
[[15952  8257]
 [ 2646  3145]]
```

[75]:
```python
from sklearn.svm import LinearSVC

# SVM using LinearSVC (faster, no probability calibration needed)
svm_clf = Pipeline(steps=[
    ("preprocess", make_preprocess()),
    ("clf", LinearSVC(
        class_weight="balanced",
        random_state=42
    ))
])

# Train
svm_clf.fit(X_train, y_train)

# Class predictions (0/1)
y_pred_svm = svm_clf.predict(X_test)
```

```
# Use decision_function scores as "probabilities" for ROC-AUC
# (roc_auc_score works with any continuous scores, not only true probabilities)
y_score_svm = svm_clf.decision_function(X_test)

# Evaluate - pass y_score_svm in place of y_proba
results.append(evaluate_model(
    "SVM (LinearSVC, decision_function)",
    y_test,
    y_pred_svm,
    y_score_svm
))
```

```
=== SVM (LinearSVC, decision_function) ===
Accuracy : 0.5976
Precision: 0.2594
Recall   : 0.5847
F1-score : 0.3594
ROC-AUC  : 0.6270
Confusion matrix:
[[14543  9666]
 [ 2405  3386]]
```

0.7 Clustering + CSE

[76]:
```
cluster_numeric_features = ["Month", "DayOfWeek", "CRSDepMinutes", "Distance"]

X_train_num = X_train[cluster_numeric_features].copy()
X_test_num = X_test[cluster_numeric_features].copy()

cluster_scaler = StandardScaler()
X_train_num_scaled = cluster_scaler.fit_transform(X_train_num)
X_test_num_scaled = cluster_scaler.transform(X_test_num)
```

[77]:
```
K = 4  # you can try 3, 4, 5 and compare

kmeans = KMeans(n_clusters=K, random_state=42, n_init=10)
train_clusters = kmeans.fit_predict(X_train_num_scaled)
test_clusters = kmeans.predict(X_test_num_scaled)

unique, counts = np.unique(train_clusters, return_counts=True)
print("Train cluster sizes:")
for u, c in zip(unique, counts):
    print(f"Cluster {u}: {c} samples")
```

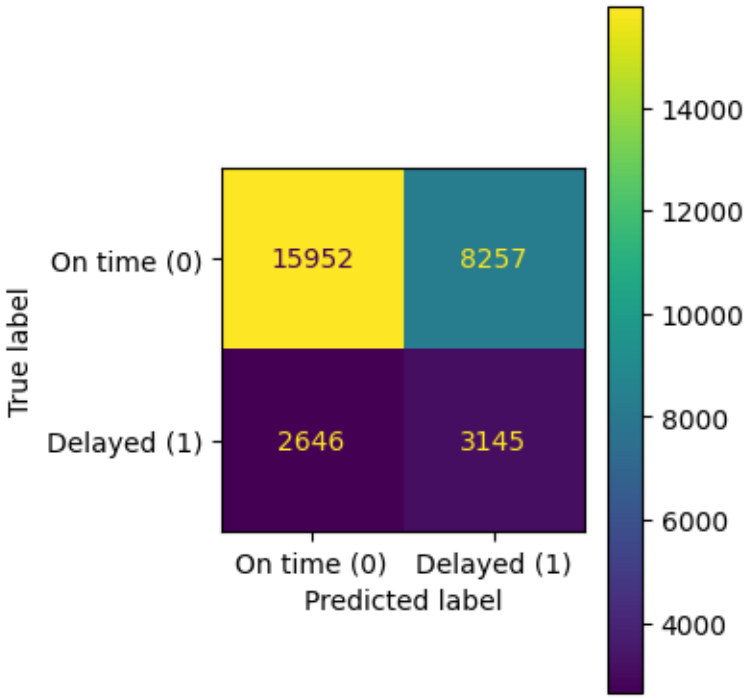
```
Train cluster sizes:
Cluster 0: 32743 samples
Cluster 1: 33530 samples
Cluster 2: 13599 samples
Cluster 3: 40128 samples
```

[78]:
```python
cluster_models = {}
MIN_CLUSTER_SIZE = 1000   # skip tiny clusters if you want

for c in range(K):
    idx = np.where(train_clusters == c)[0]
    if len(idx) < MIN_CLUSTER_SIZE:
        print(f"Skipping cluster {c} (too small: {len(idx)} samples)")
        continue

    X_train_c = X_train.iloc[idx]
    y_train_c = y_train.iloc[idx]

    print(f"Training model for cluster {c} with {len(idx)} samples")

    model_c = Pipeline(steps=[
        ("preprocess", make_preprocess()),
        ("clf", RandomForestClassifier(
            n_estimators=200,
            max_depth=15,
            class_weight="balanced_subsample",
            n_jobs=-1,
            random_state=42
        ))
    ])

    model_c.fit(X_train_c, y_train_c)
    cluster_models[c] = model_c

print("Trained cluster models:", list(cluster_models.keys()))
```

```
Training model for cluster 0 with 32743 samples
Training model for cluster 1 with 33530 samples
Training model for cluster 2 with 13599 samples
Training model for cluster 3 with 40128 samples
Trained cluster models: [0, 1, 2, 3]
```

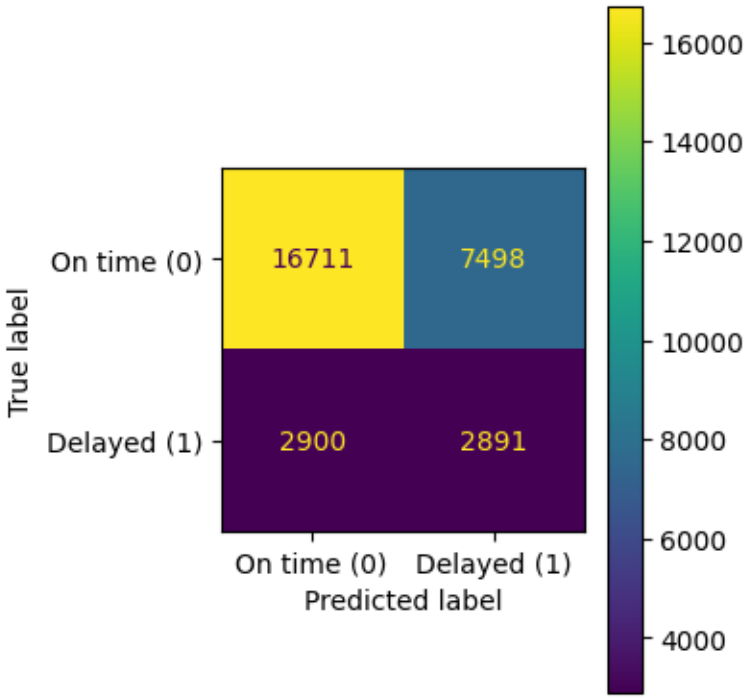
0.8 Confusion matrix plot for global Random Forest

[79]:
```
from sklearn.metrics import ConfusionMatrixDisplay


cm_rf = confusion_matrix(y_test, y_pred_rf)

disp_rf = ConfusionMatrixDisplay(
    confusion_matrix=cm_rf,
    display_labels=["On time (0)", "Delayed (1)"]
)

fig, ax = plt.subplots(figsize=(4, 4))
disp_rf.plot(ax=ax)

plt.tight_layout()
plt.show()
```



0.9 Confusion matrix plot for CSE

[80]:
```
# 0.10 Cluster-Specific Ensemble (CSE) + Confusion Matrix Plot 2

# Start from global RF predictions
y_pred_cse  = rf_clf.predict(X_test)
y_proba_cse = rf_clf.predict_proba(X_test)[:, 1]

# Override with cluster-specific models
for c, model_c in cluster_models.items():
    idx = np.where(test_clusters == c)[0]
    if len(idx) == 0:
        continue
    X_test_c = X_test.iloc[idx]
    preds_c  = model_c.predict(X_test_c)
    proba_c  = model_c.predict_proba(X_test_c)[:, 1]

    y_pred_cse[idx]  = preds_c
    y_proba_cse[idx] = proba_c

# Now plot confusion matrix
cm_cse = confusion_matrix(y_test, y_pred_cse)

disp_cse = ConfusionMatrixDisplay(
    confusion_matrix=cm_cse,
    display_labels=["On time (0)", "Delayed (1)"]
)

fig, ax = plt.subplots(figsize=(4, 4))
disp_cse.plot(ax=ax)
plt.tight_layout()
plt.show()
```



0.10 Model Performance Comparison

[81]:
```
# Start from global Random Forest predictions (fallback)
y_pred_cse = rf_clf.predict(X_test)
y_proba_cse = rf_clf.predict_proba(X_test)[:, 1]

# Override with cluster-specific models when available
for c, model_c in cluster_models.items():
    idx = np.where(test_clusters == c)[0]
    if len(idx) == 0:
        continue

    X_test_c = X_test.iloc[idx]
    preds_c = model_c.predict(X_test_c)
    proba_c = model_c.predict_proba(X_test_c)[:, 1]

    y_pred_cse[idx] = preds_c
    y_proba_cse[idx] = proba_c

# Evaluate CSE model
results.append(evaluate_model("Cluster-Specific Ensemble (CSE)", y_test,␣
 ↪y_pred_cse, y_proba_cse))
results_df = pd.DataFrame(results).set_index("model")
```

```
results_df
```

```
=== Cluster-Specific Ensemble (CSE) ===
Accuracy : 0.6534
Precision: 0.2783
Recall   : 0.4992
F1-score : 0.3574
ROC-AUC  : 0.6375
Confusion matrix:
[[16711  7498]
 [ 2900  2891]]
```

[81]:

model	accuracy	precision	recall	f1	roc_auc
Majority Class (Dummy)	0.806967	0.000000	0.000000	0.000000	NaN
Logistic Regression	0.597700	0.259168	0.583319	0.358884	0.627707
Decision Tree	0.575100	0.253508	0.617683	0.359479	0.623248
Random Forest	0.636567	0.275829	0.543084	0.365847	0.643703
SVM (LinearSVC, decision_function)	0.597633	0.259424	0.584700	0.359391	0.626968
Cluster-Specific Ensemble (CSE)	0.653400	0.278275	0.499223	0.357355	0.637490

0.11 Preprocess for neural net, build & train LSTM

[82]:
```
# Use the same preprocessing as other models, but get a dense array
preprocess_nn = make_preprocess()  # reuses your numeric + categorical pipeline

X_train_nn = preprocess_nn.fit_transform(X_train)
X_test_nn  = preprocess_nn.transform(X_test)

# If the result is sparse (because of one-hot), convert to dense
if hasattr(X_train_nn, "toarray"):
    X_train_nn = X_train_nn.toarray()
    X_test_nn  = X_test_nn.toarray()

print("Neural net feature shape:", X_train_nn.shape)
```

```
# Reshape for LSTM: (samples, timesteps, features)
# We treat each flight as a sequence of length 1 with many features
n_features = X_train_nn.shape[1]
X_train_lstm = X_train_nn.reshape(-1, 1, n_features)
X_test_lstm  = X_test_nn.reshape(-1, 1, n_features)

# Define a simple LSTM model
lstm_model = Sequential([
    LSTM(32, input_shape=(1, n_features)),
    Dense(16, activation="relu"),
    Dense(1, activation="sigmoid")
])

lstm_model.compile(
    loss="binary_crossentropy",
    optimizer=Adam(learning_rate=1e-3),
    metrics=["accuracy"]
)

lstm_model.summary()

# Train the LSTM - you can increase epochs later if it's fast
history_lstm = lstm_model.fit(
    X_train_lstm, y_train,
    validation_split=0.2,
    epochs=5,
    batch_size=256,
    verbose=1
)

# Predict on the test set
y_proba_lstm = lstm_model.predict(X_test_lstm).ravel()
y_pred_lstm  = (y_proba_lstm >= 0.5).astype(int)
```

```
Neural net feature shape: (120000, 744)
```

```
/usr/local/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 32)	99,456

```
 dense_6 (Dense)                 (None, 16)                        528

 dense_7 (Dense)                 (None, 1)                          17


 Total params: 100,001 (390.63 KB)

 Trainable params: 100,001 (390.63 KB)

 Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/5
375/375             3s 4ms/step -
accuracy: 0.7972 - loss: 0.5441 - val_accuracy: 0.8089 - val_loss: 0.4709
Epoch 2/5
375/375             1s 3ms/step -
accuracy: 0.8053 - loss: 0.4745 - val_accuracy: 0.8090 - val_loss: 0.4686
Epoch 3/5
375/375             1s 3ms/step -
accuracy: 0.8067 - loss: 0.4710 - val_accuracy: 0.8092 - val_loss: 0.4671
Epoch 4/5
375/375             1s 3ms/step -
accuracy: 0.8079 - loss: 0.4657 - val_accuracy: 0.8096 - val_loss: 0.4668
Epoch 5/5
375/375             1s 3ms/step -
accuracy: 0.8063 - loss: 0.4666 - val_accuracy: 0.8100 - val_loss: 0.4666
938/938             1s 1ms/step
```

0.12 LSTM to our results table

[83]:
```
# Add LSTM to the results list and rebuild results_df
results.append(
    evaluate_model("LSTM", y_test, y_pred_lstm, y_proba_lstm)
)

results_df = pd.DataFrame(results).set_index("model")
results_df
```

```
=== LSTM ===
Accuracy : 0.8077
Precision: 0.6146
Recall   : 0.0102
F1-score : 0.0200
```

```
ROC-AUC  : 0.6427
Confusion matrix:
[[24172    37]
 [ 5732    59]]
```

[83]:

model	accuracy	precision	recall	f1
Majority Class (Dummy)	0.806967	0.000000	0.000000	0.000000
Logistic Regression	0.597700	0.259168	0.583319	0.358884
Decision Tree	0.575100	0.253508	0.617683	0.359479
Random Forest	0.636567	0.275829	0.543084	0.365847
SVM (LinearSVC, decision_function)	0.597633	0.259424	0.584700	0.359391
Cluster-Specific Ensemble (CSE)	0.653400	0.278275	0.499223	0.357355
LSTM	0.807700	0.614583	0.010188	0.020044

model	roc_auc
Majority Class (Dummy)	NaN
Logistic Regression	0.627707
Decision Tree	0.623248
Random Forest	0.643703
SVM (LinearSVC, decision_function)	0.626968
Cluster-Specific Ensemble (CSE)	0.637490
LSTM	0.642652

[ ]: