

Author: Dr. Mallarapu

Created: 2026-07-27

Course: SEAS 8414 — Security Analytics

Goal of this notebook

Train and audit detectors on the CSE-CIC-IDS2018 brute-force day, then grade how much of the score is trustworthy.

What you will learn

1. Read a majority-class baseline before trusting any accuracy figure.
2. Find the strongest single feature, then test it by dropping it and refitting.
3. Tell duplicate inflation apart from genuine signal.
4. Report per-group recall, because the rare classes carry the risk.

Where this connects to the course text

The text builds a defence pipeline; this notebook trains a classifier and audits it. The links below are to specific chapter objectives that share an *analytic move*, not to matching subject matter.

- **Chapter 3: Vulnerability Assessment** — Learning objective 1 (section 3.1) frames assessment as **evidence grading**, not output collection. The A-F data-trust grade in section 10 is exactly that move, applied to a model score.
 - **Chapter 11: Formal Protocol Verification** — Section **11.1.2**, titled *Proved, tested, and hoped*, asks you to separate exactly those three. (Chapter 11 lists its objectives in §11.0, not §11.1 as the other chapters do.) The ablation does that job here: it tests whether the headline survives.
-

CIC-IDS2018 Case Study — FTP/SSH Brute Force

Model comparison + validity audit on the ftp/ssh brute force day of CSE-CIC-IDS2018

Abstract: We study one day of the CSE-CIC-IDS2018 flow corpus. Its attack traffic is **ftp/ssh brute force**. Benign flows still dominate the day numerically. The attack is the minority class, and the loader prints the exact rate. We load $\geq 1,000,000$ real CICFlowMeter records. The four learners then fit a **120,000-row stratified subsample**, so every score below is a subsample number. The question is whether the near-perfect

in-distribution scores reflect detection, or a defect in the features. Engelen et al. (2021) found bugs in CICFlowMeter itself. That tool built this day's features too, so their tool-level findings are a live suspicion here. Their label corrections are specific to CICIDS2017 and do not carry over to this 2018 capture.

1. Research problem

Task: Detect FTP-BruteForce and SSH-BruteForce connections among benign traffic on the 2018-02-14 capture. Brute force is high-volume and repetitive, so we expect it to be easy — and easy is exactly what makes the leakage question sharp.

2. Literature review

- **Sharafaldin, Lashkari & Ghorbani (2018)** — the CIC-IDS2017/2018 dataset paper: realistic profiled benign traffic plus a labeled attack schedule. CICFlowMeter extracts more than 80 network-traffic features. The per-day CSV used here has 80 columns, one of which is the label.
- **Engelen, Rimmer & Joosen (2021)** — *Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study* — found labeling errors and CICFlowMeter feature bugs that inflate scores.
- **Rosay, Cheval, Carlier & Leroux (2022)** — *Network Intrusion Detection: A Comprehensive Analysis of CIC-IDS2017* (ICISSP): documents CICFlowMeter implementation flaws affecting these flow features.
- **Sommer & Paxson (2010)** — closed-world ML scores rarely survive deployment.
- **Apruzzese et al. (2023)** — *The Role of Machine Learning in Cybersecurity* (ACM DTRAP). A survey of where ML is and is not actually deployed in security practice. It is cited for that framing, not as a study of dataset shortcuts.

Related approaches and their known caveats — drawn from the wider literature; these are **not** measurements reproduced on this exact corpus:

Reported approach	Known caveat
Sharafaldin et al. (2018) — RF on CIC-IDS2017/18	in-distribution; CICFlowMeter features later shown buggy
Engelen et al. (2021) — re-labeled CICIDS2017	original labels/features partly wrong
Typical DL-NIDS papers	benign-majority base rate inflates accuracy; per-family recall varies

3. Dataset provenance & honesty caveats

Property	Value
Source	CSE-CIC-IDS2018, AWS Open Data <code>s3://cse-cic-ids2018/</code> (no credentials)
File	<code>Wednesday-14-02-2018_TrafficForML_CICFlowMeter.csv</code>
Rows	1,048,575 flow records, exactly as the loader prints (the S3 download is bounded at 1.2M lines)
Features	80 columns in this CSV: 79 CICFlowMeter flow features plus <code>Label</code> . The loader drops <code>Label</code> and <code>Timestamp</code> , then drops every column with a single distinct value, and the 68 it prints is what remains. This file carries no flow-ID or IP column, so there is no identifier column to drop.
Label	<code>Benign</code> vs the day's attack families

Honestly: CICFlowMeter's feature implementation has documented bugs (Engelen et al. 2021; Rosay et al. 2022), and accuracy is inflated by the benign-majority base rate. We report per-attack-family recall and audit single-feature shortcuts for this reason. One more caveat belongs here rather than in a footnote. `Dst Port` is kept as a model feature, and this day's attacks are FTP brute force (port 21) and SSH brute force (port 22). So on this capture that column is closer to a name for the attack class than to a description of behaviour. Section 10 says what that does to the score.

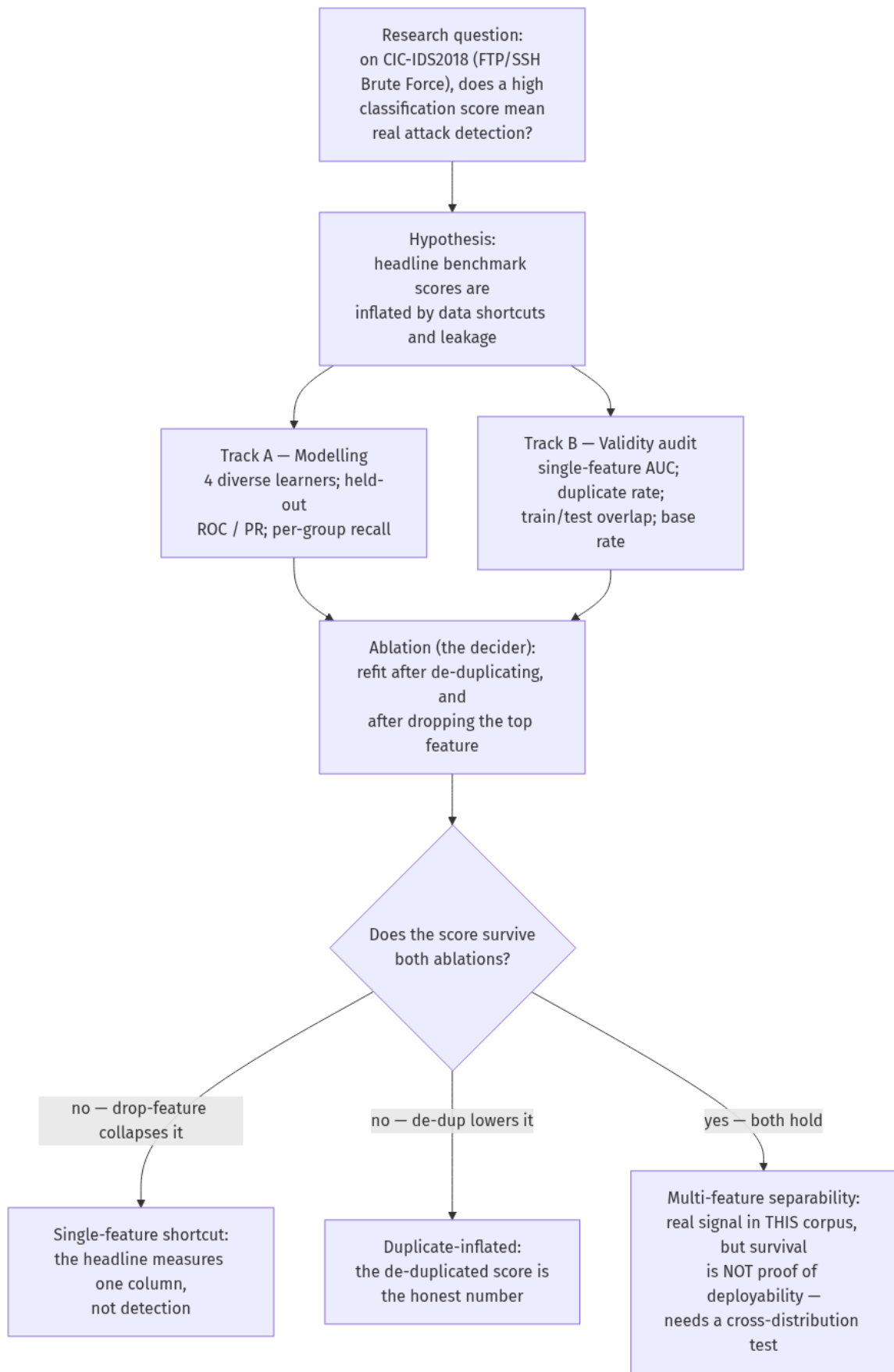
Before you run this: getting the data

This notebook downloads its own data on the first run, then caches it. **No Kaggle account and no credentials are needed** - the source is the public AWS Open Data bucket `s3://cse-cic-ids2018/`.

4. Solution design

The methodology is deliberately two-track. We *earn* a headline score with standard modelling, then *interrogate* it with a validity audit. Only a verdict that survives both is reported. The diagram below is the shape of every notebook in this series.

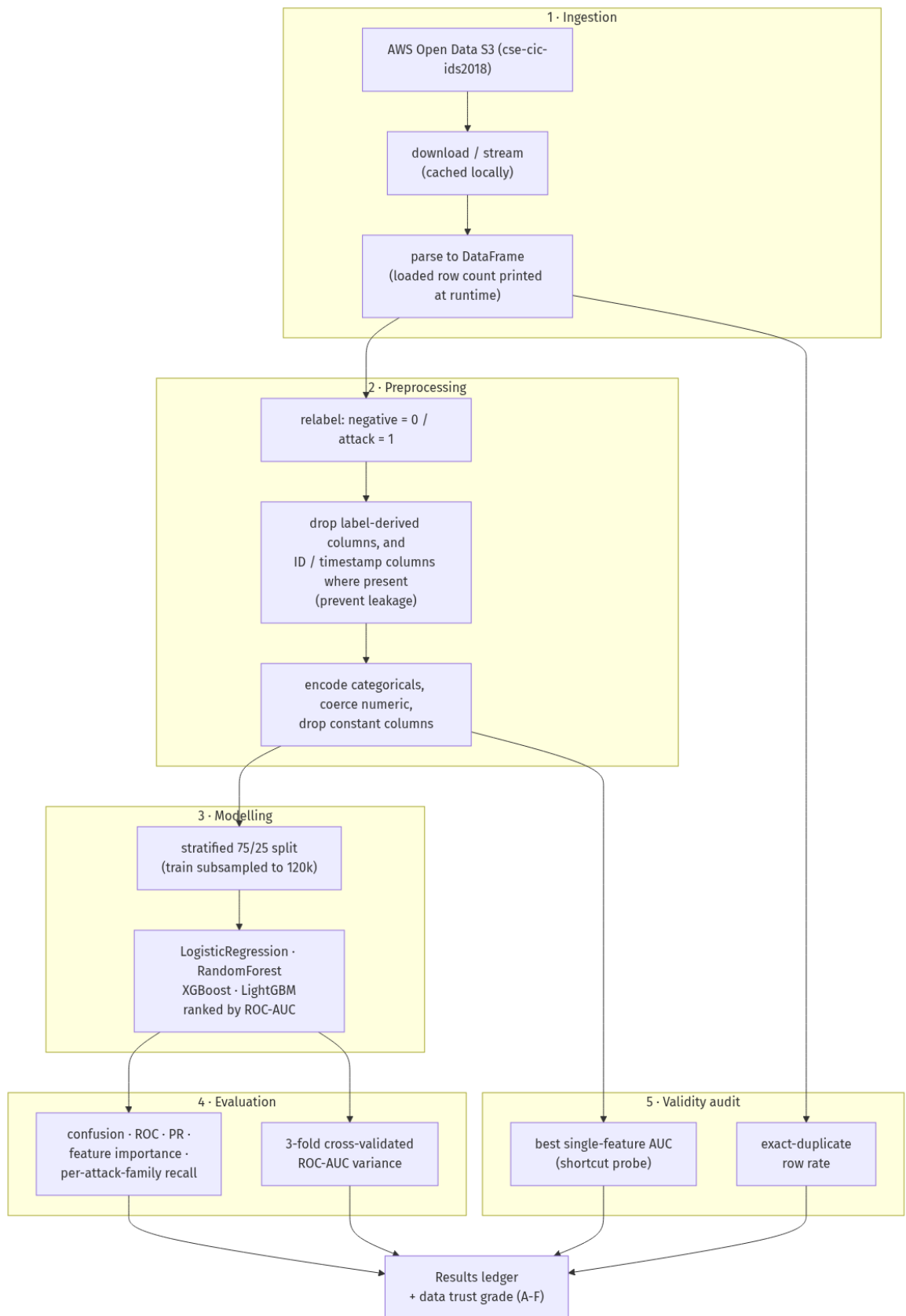
Figure 4.1 — Solution design (methodology).



5. Implementation architecture

Five stages — ingestion, preprocessing, modelling, evaluation, and a parallel validity-audit path — feed a single graded results ledger. Leakage defences (dropping label-derived and identifier columns) live in preprocessing, before any model sees the data.

Figure 5.1 — Implementation architecture.



Reading stage 2 of the figure: the "encode categoricals" step in that box is template text for this notebook series and **does not run on this corpus**. Nothing categorical is encoded here and no encoder is fitted. Every column the loader keeps is already numeric. So the loader coerces each one with `pd.to_numeric`, maps non-finite values to 0.0,

and then drops the columns with a single distinct value. The rest of the stage — relabelling, dropping the label and timestamp columns, dropping constants — is what the loader in section 6 does.

6. Data acquisition & preparation

Every line below is commented so a student can re-run and modify each step. The cell ends by producing the standard analysis variables: `df`, `X` (clean numeric features), `y` (binary label), `feat` (feature names), and `family`. `family` is the per-group label used for the recall breakdown. It is an attack family on the intrusion corpora, but a transaction type, merchant category or malware category on the fraud/malware ones.

```
In [1]: %matplotlib inline
import time, warnings; warnings.filterwarnings('ignore')    # keep output clean
import numpy as np, pandas as pd                            # numerics + dataframes
import matplotlib.pyplot as plt                             # static plots
                                                         (embed in HTML+PDF)
plt.rcParams['figure.dpi'] = 120                            # crisp figures
RANDOM_STATE = 0                                             # single seed
used everywhere
np.random.seed(RANDOM_STATE)                                # reproducible sampling
NEG_WORD, POS_WORD = 'benign', 'attack'                     # class names
                                                         (overridden by some loaders)
```

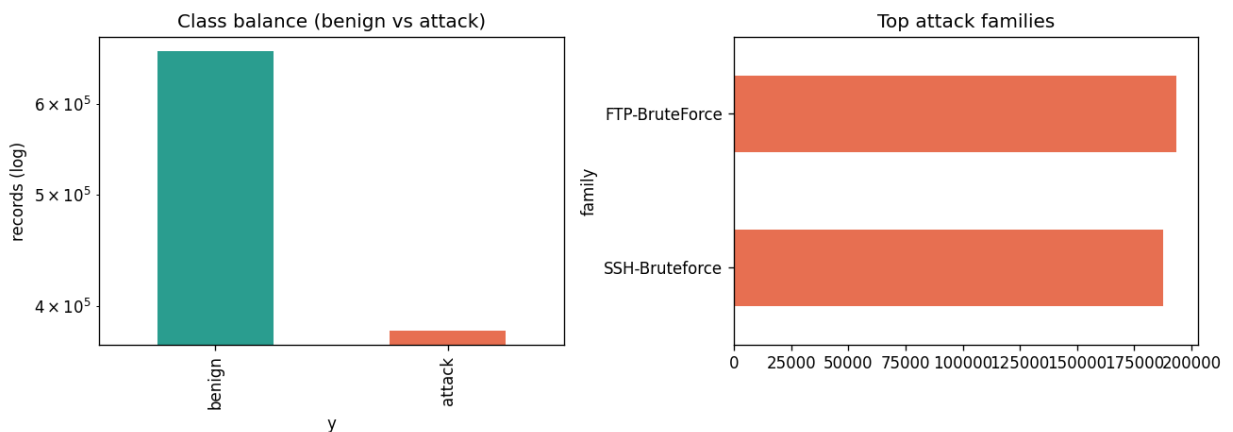
```
In [2]: import os
FILE = 'Wednesday-14-02-2018-TrafficForML-CICFlowMeter.csv'; SHORT =
'bruteforce'
PREFIX = 's3://cse-cic-ids2018/Processed Traffic Data for ML Algorithms/'
CACHE = f'/tmp/cic_{SHORT}.csv'
if not os.path.exists(CACHE):                               # bounded S3
download, no credentials
    os.system(f'aws s3 cp "{PREFIX}{FILE}" --no-sign-request 2>/dev/null
| head -n 1200000 > "{CACHE}"')
df = pd.read_csv(CACHE, low_memory=False)                   # parse the
day's flow records
df = df[pd.to_numeric(df['Dst Port'],
errors='coerce').notna()].reset_index(drop=True)           # drop repeated-header
junk
df['Label'] = df['Label'].astype(str).str.strip()            # clean labels
df['y'] = (df['Label'] != 'Benign').astype(int)              # 1 = attack,
0 = benign
df['family'] = df['Label']                                   # attack type
doubles as family
assert len(df) >= 1_000_000, f'floor not met: {len(df):,}' # honesty
gate: >= 1M rows
DROP = ['Label', 'Timestamp', 'y', 'family']
feat = [c for c in df.columns if c not in DROP]
X = df[feat].apply(pd.to_numeric, errors='coerce').replace([np.inf, -
np.inf], np.nan).fillna(0.0)
```

```
X = X.loc[:, X.nunique() > 1]; feat = list(X.columns) # drop
constants; align feat
y = df['y'].to_numpy() # STANDARD
CONTRACT: binary label
print(f'loaded {len(df):,} flows x {len(feat)} features; attack rate
{y.mean():.4f}')
```

loaded 1,048,575 flows x 68 features; attack rate 0.3633

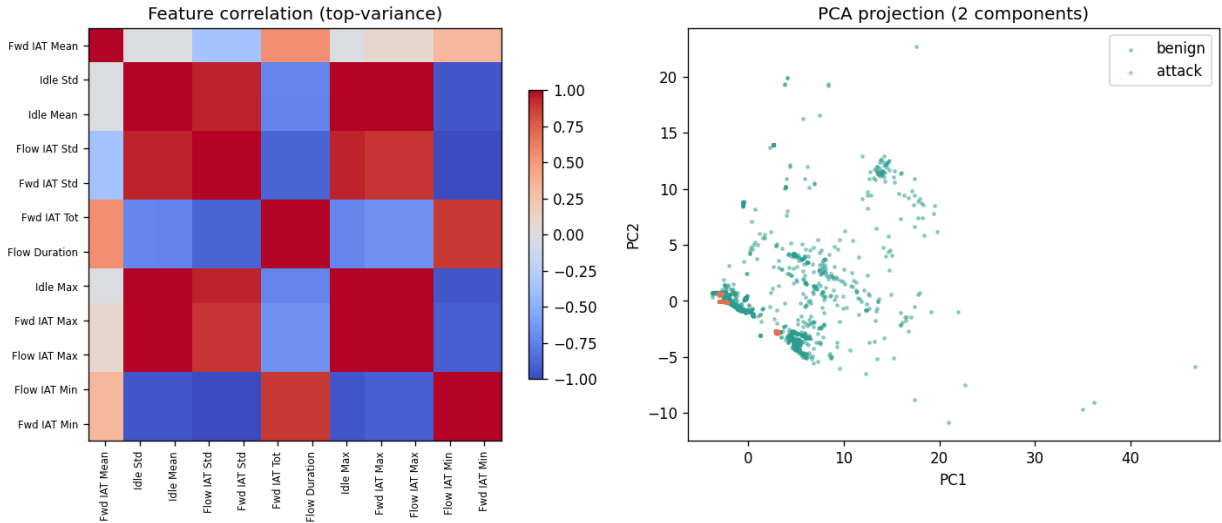
7. Exploratory data analysis

```
In [3]: # --- EDA 1: class balance and the attack-family mix ---
fig, ax = plt.subplots(1, 2, figsize=(11, 4))
df['y'].map({0:NEG_WORD,1:POS_WORD}).value_counts().plot.bar(
# counts per class
    ax=ax[0], color=['#2a9d8f','#e76f51']); ax[0].set_yscale('log')
ax[0].set_title(f'Class balance ({NEG_WORD} vs {POS_WORD})');
ax[0].set_ylabel('records (log)')
df.loc[df.y==1,'family'].value_counts().head(8).plot.barh(
# top attack families
    ax=ax[1], color='#e76f51'); ax[1].invert_yaxis(); ax[1].set_title('Top
attack families')
plt.tight_layout(); plt.show()
```



```
In [4]: # --- EDA 2: feature correlation + a 2-D PCA projection ---
from sklearn.preprocessing import StandardScaler # scale
before PCA
from sklearn.decomposition import PCA
fig, ax = plt.subplots(1, 2, figsize=(12, 5))
topv = X[feat].var().sort_values().tail(12).index # 12
highest-variance features
im = ax[0].imshow(X[topv].corr(), cmap='coolwarm', vmin=-1, vmax=1) #
correlation heatmap
ax[0].set_xticks(range(len(topv))); ax[0].set_xticklabels(topv,
rotation=90, fontsize=7)
ax[0].set_yticks(range(len(topv))); ax[0].set_yticklabels(topv, fontsize=7)
ax[0].set_title('Feature correlation (top-variance)'); fig.colorbar(im,
ax=ax[0], shrink=0.7)
samp = X.sample(min(5000, len(X)), random_state=RANDOM_STATE) #
subsample for a fast PCA
pc =
PCA(n_components=2).fit_transform(StandardScaler().fit_transform(samp))
```

```
ys = y[samp.index] # aligned
labels for coloring
for lab,c in [(0,'#2a9d8f'),(1,'#e76f51')]:
    ax[1].scatter(pc[ys==lab,0], pc[ys==lab,1], s=4, alpha=0.4, color=c,
                  label={0:NEG_WORD,1:POS_WORD}[lab])
ax[1].set_title('PCA projection (2 components)'); ax[1].legend();
ax[1].set_xlabel('PC1'); ax[1].set_ylabel('PC2')
plt.tight_layout(); plt.show()
```



8. Model comparison

Four diverse learners share one held-out split, ranked by ROC-AUC.

Two honest guards print with the table:

1. The models train on a *stratified subsample* of at most 120,000 rows. The full row count is printed above. So every score here is a subsample number, not a full-corpus claim.
2. The **majority-class baseline accuracy** appears *inside* the ranking table. On imbalanced data, 0.99 accuracy can be worse than always guessing the majority class. Judge each model against that baseline, not against 0.5.

```
In [5]: # --- Model comparison: four learners on the same held-out split ---
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
import xgboost as xgb, lightgbm as lgb

# Stratified split keeps the class ratio in both halves.
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.25,
                                       random_state=RANDOM_STATE, stratify=y)
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
N_MATERIALIZED = len(y) # the full
corpus we loaded (see printed count)
```

```

# HONEST DISCLOSURE: we do NOT train on all N. We fit on a STRATIFIED
subsample (<=120k) because
# these learners saturate long before then on this data. Every headline
below is a SUBSAMPLE
# number, not a full-corpus number – saying otherwise would be the
fabrication this course forbids.
if len(Xtr) > 120_000:
    Xtr, _, ytr, _ = train_test_split(Xtr, ytr, train_size=120_000,
                                     random_state=RANDOM_STATE,
                                     stratify=ytr)          #
genuinely stratified, not random
MAJORITY_BASELINE = max(np.mean(yte), 1 - np.mean(yte))    # accuracy
of 'always predict majority'
print(f'materialized {N_MATERIALIZED:,} rows | trained on {len(Xtr):,}
(stratified subsample) | '
      f'held-out {len(yte):,}')
print(f'MAJORITY-CLASS BASELINE accuracy = {MAJORITY_BASELINE:.4f} '
      f'(any model must beat THIS, not 0.5, to be interesting)')

models = {                                                  # four
standard, diverse learners
    'LogisticRegression': make_pipeline(StandardScaler(),
LogisticRegression(max_iter=300)), # scaled!
    'RandomForest': RandomForestClassifier(n_estimators=60, n_jobs=-1,
random_state=RANDOM_STATE),
    'XGBoost': xgb.XGBClassifier(n_estimators=80, max_depth=6,
tree_method='hist', n_jobs=-1,
                                eval_metric='logloss',
random_state=RANDOM_STATE),
    'LightGBM': lgb.LGBMClassifier(n_estimators=80, n_jobs=-1, verbose=-1,
random_state=RANDOM_STATE),
}
rows, fitted = [], {}
for name, m in models.items():                             # fit +
score each model
    t = time.perf_counter(); m.fit(Xtr, ytr); fitted[name] = m
    p = m.predict_proba(Xte)[: , 1]                        #
positive-class probability on held-out
    rows.append({'model': name, 'accuracy': round(accuracy_score(yte,
(p>0.5).astype(int)), 6),
                'roc_auc': round(roc_auc_score(yte, p), 6),    # 6 dp: a
1.000000 is a red flag, not a win
                'train_s': round(time.perf_counter()-t, 1)})
rows.append({'model': 'MajorityBaseline', 'accuracy':
round(MAJORITY_BASELINE, 4),
            'roc_auc': 0.5, 'train_s': 0.0})                # show the
baseline IN the ranking table
comparison = pd.DataFrame(rows).sort_values('roc_auc',
ascending=False).reset_index(drop=True)
_ranked = comparison[comparison.model != 'MajorityBaseline']
best_name = _ranked.iloc[0]['model']; best = fitted[best_name] # winner by
ROC-AUC (excl. baseline)
print('best model:', best_name); comparison

```

materialized 1,048,575 rows | trained on 120,000 (stratified subsample) | held-out 262,144
 MAJORITY-CLASS BASELINE accuracy = 0.6367 (any model must beat THIS, not 0.5, to be interesting)
 best model: RandomForest

Out[5]:

	model	accuracy	roc_auc	train_s
0	RandomForest	0.999992	1.000000	0.5
1	XGBoost	0.999985	1.000000	0.3
2	LightGBM	0.999969	0.999991	0.6
3	LogisticRegression	0.999874	0.999983	0.2
4	MajorityBaseline	0.636700	0.500000	0.0

9. Results

Diagnostics for the winning model, including **per-group recall**.

The grouping comes from whatever the loader put in `family`. It is *not* always an attack taxonomy. On the intrusion corpora it is the attack family. On the fraud and malware corpora it is a transaction type, a merchant category or a malware category. On binary corpora it collapses to the positive class.

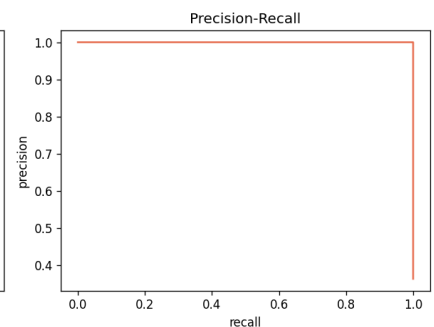
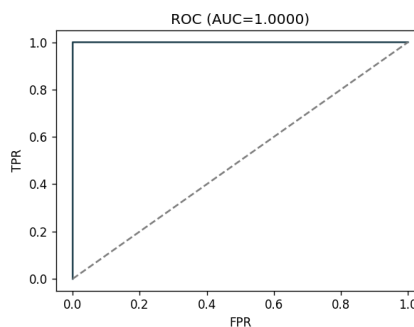
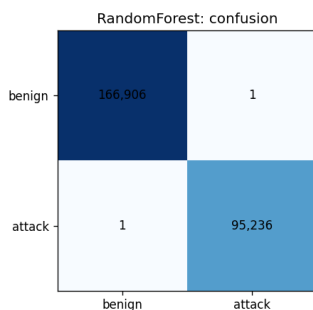
Read it accordingly. Where the groups are genuinely rare classes, they reveal whether detection is real. The dominant flood classes do not.

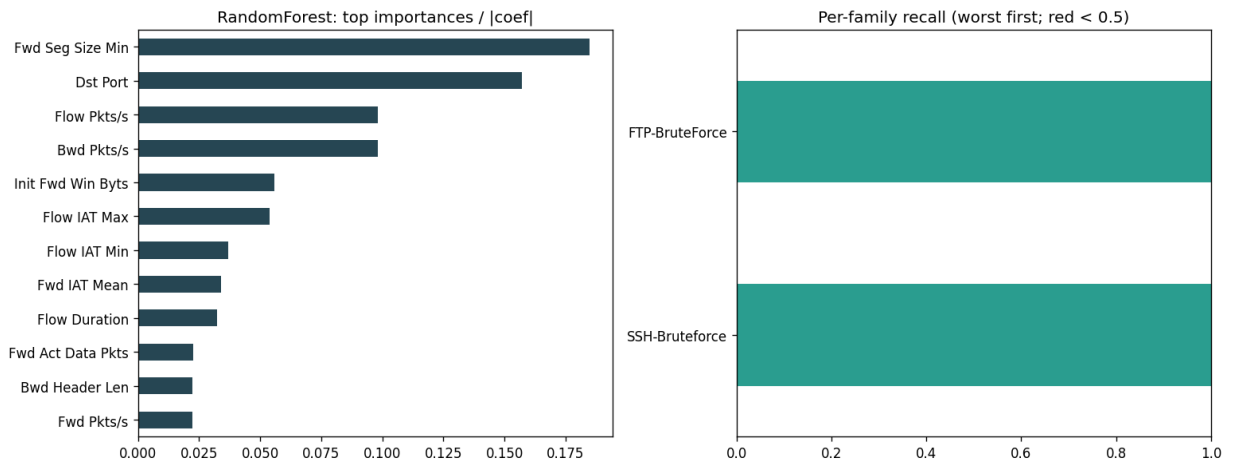
```
In [6]: # --- Results for the best model: confusion, ROC, PR, importances, per-
family recall ---
from sklearn.metrics import confusion_matrix, roc_curve,
precision_recall_curve, recall_score
pb = best.predict_proba(Xte)[: , 1]; pred = (pb > 0.5).astype(int)
fig, ax = plt.subplots(1, 3, figsize=(15, 4))
# (1) confusion matrix
cm = confusion_matrix(yte, pred); ax[0].imshow(cm, cmap='Blues')
ax[0].set_title(f'{best_name}: confusion'); ax[0].set_xticks([0,1]);
ax[0].set_yticks([0,1])
ax[0].set_xticklabels([NEG_WORD, POS_WORD]);
ax[0].set_yticklabels([NEG_WORD, POS_WORD])
for (i,j),v in np.ndenumerate(cm):
ax[0].text(j,i,f'{v: ,}', ha='center', va='center')
# (2) ROC and PR curves
fpr,tpr,_ = roc_curve(yte, pb); prec,rec,_ = precision_recall_curve(yte,
pb)
ax[1].plot(fpr,tpr,color='#264653'); ax[1].plot([0,1],[0,1], '--', c='grey')
ax[1].set_title(f'ROC (AUC={roc_auc_score(yte,pb):.4f})');
ax[1].set_xlabel('FPR'); ax[1].set_ylabel('TPR')
ax[2].plot(rec,prec,color='#e76f51'); ax[2].set_title('Precision-Recall');
ax[2].set_xlabel('recall'); ax[2].set_ylabel('precision')
plt.tight_layout(); plt.show()
```

```

# (3) feature importances + (4) per-attack-family recall
fig, ax = plt.subplots(1, 2, figsize=(13, 5))
imp, names = None, feat #
importances, robust to the scaled-LR pipeline
if hasattr(best, 'feature_importances_'): # tree
models
    imp = best.feature_importances_; names = list(getattr(best,
'feature_names_in_', feat)[:len(imp)])
elif hasattr(best, 'named_steps') and 'logisticregression' in getattr(best,
'named_steps', {}):
    imp = np.abs(best.named_steps['logisticregression'].coef_[0]); names =
feat # LR pipeline
elif hasattr(best, 'coef_'):
    imp = np.abs(best.coef_[0]); names = feat
if imp is not None:
    pd.Series(imp,
index=names[:len(imp)]).sort_values().tail(12).plot.barh(ax=ax[0],
color='#264653')
ax[0].set_title(f'{best_name}: top importances / |coef|')
# Per-family recall, WORST-first so rare, hard classes are visible, not
just the dominant floods.
fam_te = df.loc[Xte.index, 'family']
fr = {}
for fam, cnt in fam_te[YTE==1].value_counts().items():
    if cnt < 5: continue # need a
few positives for a meaningful recall
    mask = (fam_te==fam).to_numpy(); fr[fam] = recall_score(yte[mask],
pred[mask], zero_division=0)
srt = pd.Series(fr).sort_values()
show = pd.concat([srt.head(9), srt.tail(3)]) if len(srt) > 12 else srt #
worst 9 + best 3
show = show[~show.index.duplicated()]
show.plot.barh(ax=ax[1], color=['#e76f51' if v < 0.5 else '#2a9d8f' for v
in show]); ax[1].set_xlim(0,1)
ax[1].set_title('Per-family recall (worst first; red < 0.5)')
plt.tight_layout(); plt.show()
# Operational numbers, not just figures: false-positive rate and the worst
per-family recalls.
tn, fp = int(cm[0,0]), int(cm[0,1])
fpr_op = fp/(fp+tn) if (fp+tn) > 0 else float('nan') # benign
wrongly flagged @0.5
print(f'operational FALSE-POSITIVE RATE @0.5 = {fpr_op:.4f} ({fp:}, benign
flagged of {fp+tn:,})')
print('worst per-family recalls:', {k: round(v, 3) for k, v in
srt.head(6).items()})

```





operational FALSE-POSITIVE RATE @0.5 = 0.0000 (1 benign flagged of 166,907)
 worst per-family recalls: {'SSH-BruteForce': 1.0, 'FTP-BruteForce': 1.0}

10. Validity audit — is the score real?

Three diagnostics. **(a)** How well can the *single best feature*, alone, separate the classes? A near-1.0 single-feature AUC means that feature is *near-sufficient* — a shortcut (which may be legitimate signal or an artifact), not the same as target leakage. **(b)** The exact-duplicate row rate. **(c)** The **train/test exact-row contamination** — the fraction of held-out rows that are duplicates of training rows, which is what actually inflates a held-out score. The trust grade is the *worse* of the single-feature and contamination concerns.

A shortcut none of this removes: `Dst Port` stays in the feature matrix. The day's attacks are FTP brute force (port 21) and SSH brute force (port 22). Because of that, the destination port is close to an identifier of the attack class on this capture, not evidence about how the traffic behaves. The panel below ranks `Dst Port` second, behind `Fwd Seg Size Min`. The grade printed here does not account for it, and the ablation in section 11 drops only the top-ranked feature. So no number in this notebook shows what the model scores with the port column removed. Treat that as an untested part of the headline, not a cleared one.

```
In [7]: # --- Validity audit: is the score real detection, or a data shortcut? ---
from sklearn.metrics import roc_auc_score
samp = X.sample(min(60_000, len(X)), random_state=1); ysamp = y[samp.index]
aucs = {}
for c in feat:                                     # AUC of
    EACH feature alone
        col = samp[c].to_numpy(float)
        if col.std()==0: continue
        a = roc_auc_score(ysamp, col); aucs[c] = max(a, 1-a)      #
direction-agnostic
best_auc = max(aucs.values()); best_col = max(aucs, key=aucs.get)
dup_rate = 1 - X.drop_duplicates().shape[0]/len(X)             # exact-
duplicate feature rows (whole set)
# The statistic that actually inflates a held-out score is TRAIN/TEST
CONTAMINATION: how many test
# rows are exact duplicates of a training row. Measure it directly on the
```

```

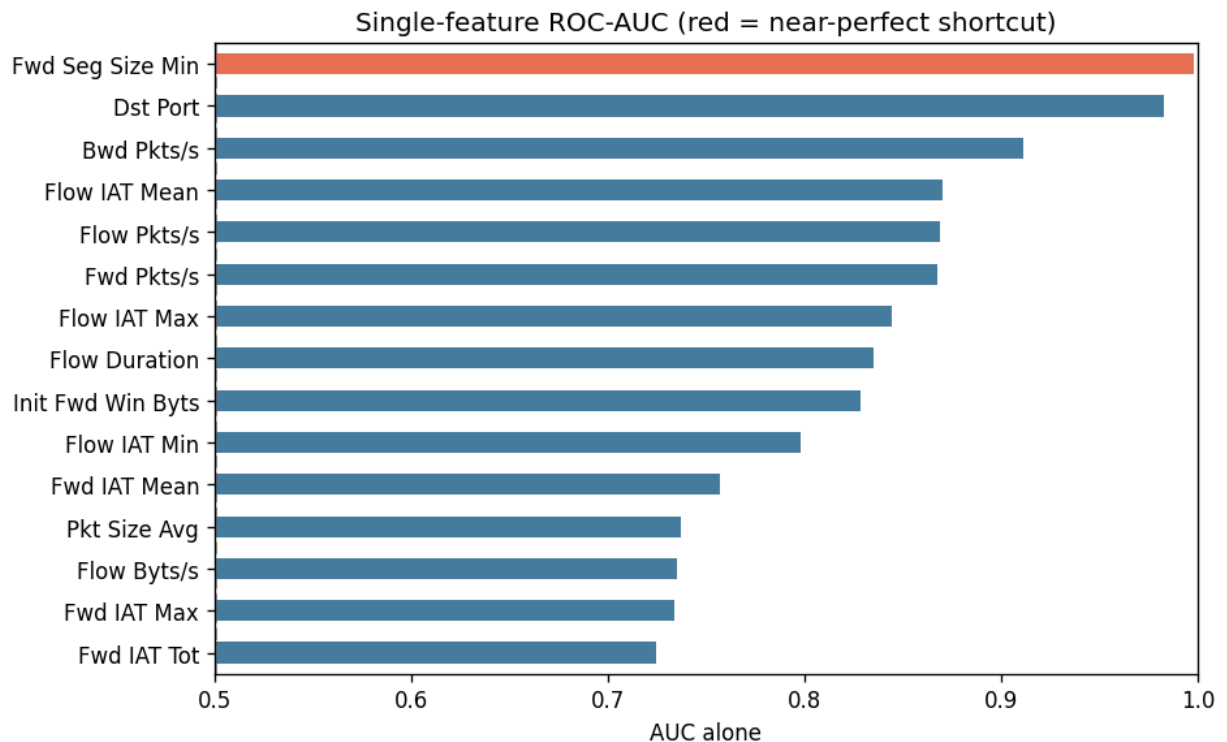
split used above.
_trkeys = set(map(tuple, np.round(Xtr.to_numpy(), 6)))
_te = np.round(Xte.to_numpy(), 6)[:50_000]
contam = float(np.mean([tuple(r) in _trkeys for r in _te]))      # fraction
of test rows seen in train
# Trust grade reflects BOTH failure modes and takes the WORSE of the two: a
near-perfect single
# feature (shortcut) OR heavy train/test contamination each independently
invalidate the headline.
_ga = 'F' if best_auc>=0.999 else 'D' if best_auc>=0.99 else 'C' if
best_auc>=0.95 else 'B' if best_auc>=0.85 else 'A'
_gc = 'F' if contam>=0.5 else 'D' if contam>=0.3 else 'C' if contam>=0.15
else 'B' if contam>=0.05 else 'A'
grade = max(_ga, _gc)                                          # 'max'
letter = worse grade (A best, F worst)
print(f'best single-feature AUC = {best_auc:.4f} (feature: {best_col})')
print(f'  note: a near-1.0 single-feature AUC means this feature is *near-
sufficient* (a shortcut),\n'
      f'  which may be legitimate signal OR an artifact – it is NOT the
same as target leakage.')
print(f'exact-duplicate row rate (whole corpus) = {dup_rate:.3f}')
print(f'TRAIN/TEST exact-row contamination      = {contam:.3f} (single-
feat grade {_ga}, contam grade {_gc})')
print(f'==> data trust grade: {grade} (worse of the two; F = shortcut
and/or heavy contamination)')
s = pd.Series(aucs).sort_values().tail(15)
fig, ax = plt.subplots(figsize=(8,5))
s.plot.barh(ax=ax, color=['#e76f51' if v>=0.99 else '#457b9d' for v in s]);
ax.axvline(0.5,ls='--',c='grey')
ax.set_xlim(0.5,1.0); ax.set_title('Single-feature ROC-AUC (red = near-
perfect shortcut)'); ax.set_xlabel('AUC alone')
plt.tight_layout(); plt.show()

```

```

best single-feature AUC = 0.9984 (feature: Fwd Seg Size Min)
  note: a near-1.0 single-feature AUC means this feature is *near-
sufficient* (a shortcut),
  which may be legitimate signal OR an artifact – it is NOT the same as
target leakage.
exact-duplicate row rate (whole corpus) = 0.356
TRAIN/TEST exact-row contamination      = 0.329 (single-feat grade D,
contam grade D)
==> data trust grade: D (worse of the two; F = shortcut and/or heavy
contamination)

```



11. Ablation — does the headline survive removing the artifacts?

Narrating a shortcut is not enough. We *retrain the winning model* after (1) de-duplicating the corpus (removing the train/test contamination) and (2) dropping the single strongest feature. We report the held-out AUC each time. **Read the result honestly, both ways:** if the AUC **collapses**, the headline was a contamination/shortcut artifact. If it **barely moves** — common on *simulated* corpora — that is **not vindication**. It means the classes are separable by *many* redundant features, because the attack and benign distributions barely overlap. That is its own generation artifact. The numbers below decide which story is true here, not the prose.

```
In [8]: # --- Ablation: SHOW the inflation empirically, don't just narrate it ---
from sklearn.base import clone
def _retrain_auc(Xa, ya):                                     # re-
    split, stratified-subsample, refit best family
    xtr, xte, ytr2, yte2 = train_test_split(Xa, ya, test_size=0.25,
    random_state=RANDOM_STATE, stratify=ya)
    if len(xtr) > 120_000:
        xtr, _, ytr2, _ = train_test_split(xtr, ytr2, train_size=120_000,
        random_state=RANDOM_STATE, stratify=ytr2)
        m = clone(best); m.fit(xtr, ytr2)
        return roc_auc_score(yte2, m.predict_proba(xte)[: , 1])
    base_auc = roc_auc_score(yte, best.predict_proba(Xte)[: , 1]) # (0) the
    headline held-out AUC
    Xdd = X.drop_duplicates(); ydd = y[Xdd.index]                # (1) de-
    duplicated corpus
    auc_dedup = _retrain_auc(Xdd, ydd)
```

```

auc_noshort = _retrain_auc(X.drop(columns=[best_col]), y) if best_col in
X.columns else base_auc # (2) drop shortcut
ablation = pd.DataFrame([
    {'setting': 'headline (as-is)', 'held_out_auc':
round(base_auc, 6)},
    {'setting': f'de-duplicated ({1-len(Xdd)/len(X):.0%} rows removed)',
'held_out_auc': round(auc_dedup, 6)},
    {'setting': f'shortcut feature dropped ({best_col})', 'held_out_auc':
round(auc_noshort, 6)},
])
print('Ablation – how much of the headline survives once each artifact is
removed:')
ablation

```

Ablation – how much of the headline survives once each artifact is removed:

```

Out[8]:

```

	setting	held_out_auc
0	headline (as-is)	1.0
1	de-duplicated (36% rows removed)	1.0
2	shortcut feature dropped (Fwd Seg Size Min)	1.0

12. Reproducibility & robustness

```

In [9]: # --- Reproducibility & robustness ---
import sklearn
from sklearn.model_selection import StratifiedKFold, cross_val_score
print(f'seed={RANDOM_STATE} | numpy {np.__version__} | sklearn
{sklearn.__version__} | '
      f'xgboost {xgb.__version__} | lightgbm {lgb.__version__}')
# 3-fold cross-validated ROC-AUC of the winning model (fresh clone, bounded
subsample) -> mean +/- std.
from sklearn.base import clone
cvX, cvy = Xtr.iloc[:40_000], ytr[:40_000]
def _auc_scorer(est, Xv, yv): # robust to
xgboost's 2-col predict_proba
    p = est.predict_proba(Xv)
    p = p[:, 1] if getattr(p, 'ndim', 1) == 2 else p
    return roc_auc_score(yv, p)
try:
    cv = cross_val_score(clone(best), cvX, cvy,
                        cv=StratifiedKFold(3, shuffle=True,
random_state=RANDOM_STATE),
                        scoring=_auc_scorer, error_score='raise')
    assert np.all(np.isfinite(cv)), 'non-finite CV folds' # FAIL CLOSED:
never narrate a NaN as evidence
    print(f'{best_name} 3-fold CV ROC-AUC = {cv.mean():.4f} +/-
{cv.std():.4f} '
          f'(mean +/- std across 3 stratified folds; a small std means a
stable estimate on this split)')
except Exception as e:
    print(f'CV UNAVAILABLE ({type(e).__name__}: {str(e)[:60]}); rely on the

```

```
single held-out AUC above — '  
f'we do NOT report a CV number we could not compute')
```

```
seed=0 | numpy 2.3.5 | sklearn 1.9.0 | xgboost 1.6.2 | lightgbm 4.7.0  
RandomForest 3-fold CV ROC-AUC = 1.0000 +/- 0.0000 (mean +/- std across 3  
stratified folds; a small std means a stable estimate on this split)
```

13. Scientific conclusion

Four learners reach near-perfect ROC-AUC on this day's flows. The ledger below names the actual strongest single feature and reports what happens when it is removed — read that before assuming a service-fingerprint story. What this capture certainly shows is that brute-force traffic is trivially separable *in this recording*. Whether that reflects attack behaviour or how CSE-CIC-IDS2018 was generated is precisely what the CICFlowMeter-defect literature disputes (Engelen et al., 2021; Rosay et al., 2022). One in-distribution score cannot settle it.

Validity ledger — read the headline against these printed numbers: Majority-class baseline **accuracy: 0.6367**. The accuracy column must clear that bar to mean anything. For ROC-AUC the trivial baseline is 0.5, not that figure. Winning learner: **RandomForest** (3-fold CV ROC-AUC **1.0000**). Strongest *single* feature: **Fwd Seg Size Min** at AUC **0.9984**. The ablation refutes a **Fwd Seg Size Min** story. Dropping that feature barely moves the AUC: **1.000000 → 1.000000**. So the separability is **multi-feature**. That reflects how this corpus was generated rather than that one column. Read *multi-feature* narrowly, though. **Dst Port** was kept as a feature and was dropped in no ablation. This day's attacks are FTP brute force (port 21) and SSH brute force (port 22). On such a day, **Dst Port** is close to a name for the attack class. So part of the surviving separability may be the port rather than behaviour. The single-feature panel ranks it second, behind **Fwd Seg Size Min**. De-duplication does **not** lower the score (**1.000000**). So duplicate rows are not what props it up. The overlap above is a caveat about the *random split*, not proof the score is fabricated. Data-trust grade: **D**. It is the worse of two independent sub-checks. Single-feature AUC 0.9984 scores **D**. Train/test exact-row overlap 0.329 scores **D**. Both sub-checks land on the same grade. On this A-best / F-worst scale, a D or F means the headline is optimistic. Treat it as a benchmark number, not a deployment estimate. Operational false-positive rate at threshold 0.5: **0.0000**. Worst per-group recalls, exactly as printed: { **SSH-Bruteforce** : 1.0, **FTP-BruteForce** : 1.0}. Every group listed is recovered essentially perfectly. So there is no rare-class failure to report on this split. That uniformity suggests the corpus is easy to separate, not that the detector is strong. **How the audit numbers are computed:** overlap is measured on the first 50,000 held-out rows, so read it as a sampled estimate. Each ablation re-splits and refits, so tiny differences are re-split noise. The de-duplication variant keeps the first label when a feature vector appears twice. **Scope:** the split is random, not temporal or entity-grouped. Every number above therefore measures in-distribution separability only.

References

1. Sharafaldin, I., Lashkari, A.H. & Ghorbani, A.A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *ICISSP*.
2. Engelen, G., Rimmer, V. & Joosen, W. (2021). Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study. *IEEE S&P Workshops*.
3. Rosay, A., Cheval, E., Carlier, F. & Leroux, P. (2022). Network Intrusion Detection: A Comprehensive Analysis of CIC-IDS2017. *8th Int. Conf. on Information Systems Security and Privacy (ICISSP)*, 25–36.
4. Sommer, R. & Paxson, V. (2010). Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. *IEEE S&P*.
5. Apruzzese, G. et al. (2023). The role of machine learning in cybersecurity. *ACM DTRAP*.