# CSCA5622_Final_Project_Nathan_Garza

September 29, 2025

# 1 CSCA5622 Supervised Learning Final Project

## 1.1 Table of Contents

  - Iteration 2 - Decision view: confusion matrix at the locked threshold and an interpretive average-return summary by predicted class.
  - Model iteration comparison — Iteration 1 (original) vs Iteration 2 (extensions)
- Results and Analysis
  - Interpretation and comparisons
  - Final interpretation
- Discussion and Conclusion

## 1.2 Project Topic - Part A

Is there a clear explanation of what this project is about? Does it state clearly which type of problem? E.g. type of learning and type of the task.

(3 pts) Gives a clear explanation of what the project is about and clearly states both the type of learning/algorithms and type of task.

### 1.2.1 Project Topic — Part A: Predicting Post–Insider-Trade Stock Direction Using Supervised Learning

I am framing this project as a supervised machine learning problem that uses two data sources I already have locally: (1) reported insider transactions from SEC Form 4 filings (the `insider_trades` dataset) and (2) daily price/volume bars for the full market (the `ohlc_1d_full_market` dataset). Concretely, I will study whether open-market insider purchases contain short-horizon predictive information for future stock direction. I will do this by merging insider trade events to daily OHLCV data and training a classifier to predict if the stock will have a positive forward return over the next k trading days (I will start with k = 20 and will compare k = 5 and k = 10 in sensitivity checks to capture shorter information-decay horizons).

- I chose insider purchases specifically (SEC transaction code `P`) because they represent a clear, economically intuitive signal: an insider spending their own cash to buy shares on the open market. Other codes (e.g., grants `A`, option exercises `M`, dispositions `S`) can be confounded by compensation structures or tax planning. I can still keep the other codes as features or run ablation analyses, but my primary label will be anchored on purchases.
- I will define each labeled example at the market-disclosure event level to reflect when information becomes public. Concretely, I will anchor the event to the Form 4 filing acceptance timestamp and set T0 = the first trading day after the filing is accepted by the SEC. I will compute the forward return from close on T0 through close at T0+k and binarize it: positive ( 0) vs. negative (< 0). This makes the task a straightforward binary classification problem. If a filing acceptance timestamp is missing, I will fall back to the transaction date and again use the first trading day after that date. I will also run a robustness check anchoring T0 to the transaction date to test whether results depend on disclosure timing versus trade timing.
- I will handle multiple purchases that occur for the same ticker and the same disclosure event by consolidating them into a single market-event row. Specifically, I will group all `P` purchase lines that share the same ticker and Form 4 acceptance timestamp, summing shares and dollar amounts, computing a volume-weighted average price, and recording the number of distinct insiders and total number of purchase lines as features. This aggregation aligns with how the market observes the disclosure (as one event) while preserving intensity information. As a sensitivity analysis, I can also treat each filer as a separate row.
- I will engineer features that are available strictly before the label window begins to avoid

look-ahead leakage. For OHLCV-based technicals (e.g., momentum, volatility, liquidity, gaps, candle ranges), I will compute them using data through the last close before T0. For insider descriptors (e.g., `officer_title`, `is_director`, `is_officer`, `is_ten_percent_owner`, `natureofownership`, transaction dollar size, relative size vs. prior holdings, direct vs. indirect), I will use only information available from the filing itself or prior history.

I am taking this approach because it is both practically relevant (investors care about whether insider buying “works”) and scientifically grounded: it allows me to test the incremental predictive content of insider activity beyond price/volume context. The one-year horizon in my data is sufficient to prototype and evaluate models using time-aware validation while carefully guarding against temporal leakage.

### 1.2.2 Type of learning and algorithms I will use

- This is a supervised learning setup because I construct explicit input–output pairs: features observed at the time of the disclosure event (inputs) and a known future outcome (the sign of the k-day forward return; the label) derived from OHLCV.
- I will focus strictly on classical supervised learning methods (no deep learning, no unsupervised modeling). My primary algorithms will be:
    - Logistic Regression with L2 regularization (baseline, interpretable coefficients, probability outputs).
    - Linear Support Vector Classifier (hinge-loss, margin-based linear model; robust baseline for high-dimensional sparse encodings).
    - Random Forest Classifier (nonlinear interactions and robust to monotonic transformations; simple feature importance).
    - Gradient Boosting Classifier (e.g., scikit-learn’s `GradientBoostingClassifier`) for stronger nonlinear modeling and calibrated probabilities.
- I will use time-series-aware cross-validation to respect temporal order. Specifically, I will start with an expanding `TimeSeriesSplit` and include a 1-day gap between the end of training and the start of validation windows to prevent leakage from contemporaneous moves around T0. I will run a rolling-window variant as a sensitivity check.
- I will standardize/encode features as appropriate per model (e.g., scaling for linear models; one-hot encoding for categorical insider attributes) using pipelines to avoid leakage.
- I will address class imbalance explicitly. For linear models I will begin with `class_weight='balanced'`. I will sweep decision thresholds based on validation PR curves and report metrics that are insensitive to prevalence (ROC-AUC, PR-AUC). If needed, I will calibrate probabilities using `CalibratedClassifierCV` (isotonic or Platt scaling) fit on time-based folds.

### 1.2.3 Type of task I am solving

- This is a binary classification task: predict whether the k-day forward return after an insider disclosure event is non-negative (class 1) or negative (class 0).
- Target construction (for ticker T and disclosure-anchored event at T0):
    - Compute forward return r_k = Close(T, T0+k) / Close(T, T0) − 1 using daily OHLCV.
    - Label y = 1 if r_k   0; else y = 0.
- Evaluation will emphasize ranking/discrimination and decision utility: ROC-AUC, PR-AUC (if classes are imbalanced), and calibration curves. I will also report more tangible finance-adjacent summaries (e.g., average forward return within predicted-positive vs. pre-

dicted-negative groups) for interpretability, while keeping the ML focus squarely on supervised classification. I will select operating thresholds using validation data (e.g., Youden's J or F1) and then lock them for final evaluation.

#### 1.2.4 Feature-freeze timing (to avoid leakage)

I freeze features before the label window and anchor events to disclosure timing:

| Item | Definition |
|---|---|
| Anchor T0 | First trading day after Form 4 filing acceptance (t_file). |
| Alternate anchor (robustness) | T0  = first trading day after transaction date (t_tran). |
| Feature window | I use data through the last close before T0 (i.e., T0−1). |
| Label window | I use closes from T0 to T0 + k. |

#### 1.2.5 My thought process and design choices

- I am choosing classification (direction) over regression (magnitude) as the primary task because directional accuracy is easier to evaluate robustly under limited samples and varying volatility regimes. I can optionally include a secondary regression analysis for magnitude as an extension, but it is not required for my supervised classification deliverable.
- I am starting with the cleanest economic subset (open-market `P` purchases) to reduce label noise. If time permits, I may layer in other event types as covariates or explore stratified models by insider role.
- To avoid leakage, I will ensure that all engineered features are computed using only information known up to and including the last close before T0 (i.e., T0−1). Validation will be walk-forward with a temporal gap to respect time.
- The one-year span should still allow multiple splits (e.g., train on early months, validate on later months). I will prioritize simpler, well-regularized models first to establish strong baselines before moving to nonlinear ensembles.

In short, I am proposing a supervised binary classification project that predicts the short-horizon direction of stock returns following insider purchases by combining insider-specific descriptors with recent OHLCV-based technical context. I will implement and compare several classical supervised algorithms (logistic regression, linear SVM, random forest, gradient boosting), evaluate them with time-aware validation and appropriate classification metrics, and discuss the strengths and limitations of each within this clearly defined supervised learning framework.

### 1.3 Project Topic - Part B

Is the goal of the project clearly stated? E.g. why it's important, what goal the author wants to achieve, or wants to learn.

(2 pts) Very Good — clearly states the motivation or the goal for the project

### 1.3.1 Project Topic — Part B: Goals and Motivation

I want to rigorously answer a focused question: after an insider open-market purchase (SEC code `P`), can I predict whether the stock's short-horizon forward return (e.g., 20 trading days) will be non-negative? This matters because insider purchases are widely cited as "smart money" signals in practice, yet the strength, stability, and conditional drivers of that signal are often anecdotal. By treating this as a supervised classification problem, I am able to define a clear label (the sign of the k-day forward return), build features known at decision time (event and OHLCV context), and evaluate out-of-sample performance in a time-aware manner. In this section I explicitly align timing with Part A: I anchor each event at T0 = the first trading day after the Form 4 filing acceptance, I compute features using data available through the last close before T0 (i.e., T0−1), and I define the k-day label from close at T0 to close at T0+k.

I am motivated by three angles: - Practical relevance: Investors, portfolio managers, and even corporate governance researchers care about whether insider buying conveys incremental information beyond price/volume history. If a consistent, risk-aware directional edge exists, it can inform screening, research triage, or post-event monitoring. If it does not, that finding is also valuable, cautioning against over-interpreting insider events. - Scientific clarity: Because Form 4 filings are standardized and OHLCV is ubiquitous, I can define precise input–output pairs and avoid common pitfalls like label leakage. This lets me test a concrete hypothesis with controlled, reproducible methodology rather than loose narratives. - Personal learning: I want to strengthen my applied supervised ML workflow on event-driven financial data: proper label construction, leakage control, time-aware cross-validation, calibration, handling class imbalance, and interpreting models that combine categorical insider descriptors with continuous technical features.

My primary goal is to build and compare several classical supervised classifiers (logistic regression, linear SVM, random forest, gradient boosting) that use only information available up to the last close before T0, and to assess whether any model demonstrates robust, out-of-sample discrimination between positive and negative k-day outcomes. I also want to produce well-calibrated probabilities so I can translate predictions into thresholded decisions and assess decision utility. For decision rules, I will pre-commit to selecting a probability threshold that maximizes F1 on validation folds and then lock this threshold before any final evaluation.

What I want to learn and demonstrate in depth: - How much incremental predictive value insider attributes add on top of recent price/volume context. For example, does officer seniority, direct vs. indirect ownership, or relative transaction size matter after controlling for momentum and volatility? - Which features consistently drive predictions, via interpretable coefficients (logistic), impurity-based or permutation importance (trees/ensembles), and partial-dependence-style diagnostics where appropriate. - How sensitive results are to horizon (k = 5, 10, 20), market regime (early vs. late in the year), and sector/size buckets, using strictly time-respecting validation. - How to balance discrimination (ROC-AUC, PR-AUC) with calibration (Brier score, reliability curves) for a decision-making context.

Why this is important in the course context: - It is a clean, supervised setup that forces me to apply best practices taught in class: careful target definition, feature engineering, regularization, cross-validation that respects temporal order, evaluation with multiple complementary metrics, and transparent model comparison. - It showcases the difference between correlation in-sample and true, temporally consistent predictability out-of-sample—a key lesson in supervised learning with time-indexed data.

Success criteria I will hold myself to: - Primary metric: ROC-AUC on held-out, time-respecting validation folds. Secondary metrics: PR-AUC, Brier score, and calibration (reliability curves). - Baselines: I will benchmark against (i) a non-informative always-positive classifier, (ii) a class-ratio probabilistic baseline, and (iii) a simple technical baseline (k-day momentum sign). I will target at least +0.05 ROC-AUC over the best baseline. - Threshold rule: I will choose a probability cutoff by maximizing F1 on validation folds and then keep that cutoff fixed for final reporting. - Uncertainty: I will report mean ± 95% confidence intervals across walk-forward splits (and show per-split metrics) to convey stability. - Decision utility: I will summarize average forward returns of predicted-positive vs. predicted-negative cohorts, including variants net of a simple friction assumption (e.g., 10 bps per trade). These summaries are for interpretability only and are not presented as trading backtests. - Reproducibility: I will fix random seeds (`random_state`) for all models, record package versions, implement preprocessing via scikit-learn Pipelines to prevent leakage, and save the exact train/validation split definitions.

Scope and constraints I acknowledge: - I am working with one year of data, which limits sample size and regime diversity. I will mitigate this by using conservative validation, focusing on interpretable models first, and clearly communicating uncertainty. - I will strictly avoid look-ahead by computing features only from data available up to the last close before T0 and validating strictly in chronological order.

In short, my goal is to produce a transparent, well-validated supervised classifier that quantifies whether insider purchase events carry short-term directional information beyond basic technical context, and to distill practical, model-agnostic insights about which insider and market features (if any) are most informative. This gives me a useful, real-world testbed for the supervised learning methods from this course and a concrete, defensible answer to an oft-debated question in equity markets.

## 1.4 Data - Part A

Is the data source properly cited and described? (including links, brief explanations)

(1 pts) Includes a brief explanation of where the data is from/how it was gathered and if the data is from a public source, cites the dataset using the format of a style manual like APA.

### 1.4.1 Data — Part A: Sources, Collection, and Citations

I purchased both datasets directly from Unusual Whales (`unusualwhales.com`). Because high-quality financial data is expensive, I was only able to afford one year of coverage. I downloaded the data as Parquet files and saved them in this project under the `data/` directory. Where relevant, I also reference the SEC's official description of Form 4 transaction codes.

#### Where I obtained the data and how it was gathered

- I acquired two datasets from Unusual Whales: daily OHLCV bars for the full market and reported insider trades (Form 4 filings).
- The OHLC daily dataset aggregates market trading into one row per trading day and ticker with fields for open, high, low, close, and volume.
- The insider trades dataset is derived from SEC Form 4 filings and includes the filing/transaction dates, roles (e.g., officer, director, 10% owner), and transaction metadata

(codes, prices, holdings context). Unusual Whales parses and distributes this information in a consistent tabular format.
- The exact coverage window for both datasets is 2024-08-29 through 2025-08-28 (inclusive).
- I stored both datasets locally in Parquet format for efficient I/O.

**Dataset overview**

| Dataset | Source | Timeframe | Format | Contents | Local path (project) |
|---|---|---|---|---|---|
| OHLC Daily Data | Unusual Whales | 1 year | Parquet | Daily open, high, low, close, volume for all market tickers | `data/ohlc_1d_full_market/` |
| Insider Trades Data | Unusual Whales | 1 year | Parquet | Reported insider trades based on SEC Form 4 filings | `data/insider_trades/1y/20` |

---

**OHLC Daily Data — field labels and descriptions**

| Field Label | Description |
|---|---|
| `ticker` | the ticker symbol for the respective security |
| `open` | the opening price of the daily candle |
| `high` | the high price of the daily candle |
| `low` | the low price of the daily candle |
| `close` | the closing price of the daily candle |
| `volume` | the total volume of the daily candle |
| `date` | the date of the daily candle |

---

**Insider Trades Data — field labels and descriptions**

| Field Label | Description |
|---|---|
| `ticker` | the ticker symbol for the respective security |
| `filing_date` | the date the trade was publicly recorded |
| `transaction_date` | the date the security transaction occurred |
| `formtype` | type of form submitted to the SEC |
| `owner_name` | name of the security owner |
| `officer_title` | professional work title |
| `is_director` | True/False if the owner is a director |
| `is_officer` | True/False if the owner is an officer |

| Field Label | Description |
|---|---|
| `is_ten_percent_owner` | True/False if the owner owns 10% or more |
| `security_ad_code` | vendor-provided security code field (as supplied) |
| `transaction_code` | SEC code indicating the type of transaction |
| `shares_owned_before` | number of shares owned prior to this transaction |
| `price` | the security price for the transaction |
| `security_title` | title of the security |
| `director_indirect` | True/False if security was purchased directly by owner (direct vs. indirect) |
| `natureofownership` | the legal nature of ownership (e.g., by trust) |
| `date_excercisable` | the date a security option contract becomes exercisable |
| `expiration_date` | the expiration date of a security option contract |
| `unique_identifier` | a unique id for this trade |

---

**SEC Form 4 transaction codes**

| Category | Code | Meaning |
|---|---|---|
| General | `P` | Open market or private purchase of non-derivative or derivative security |
| General | `S` | Open market or private sale of non-derivative or derivative security |
| General | `V` | Transaction voluntarily reported earlier than required |
| Rule 16b-3 | `A` | Grant, award or other acquisition pursuant to Rule 16b-3(d) |
| Rule 16b-3 | `D` | Disposition to the issuer of issuer equity securities pursuant to Rule 16b-3(e) |
| Rule 16b-3 | `F` | Payment of exercise price or tax liability by delivering/withholding securities incident to receipt/exercise/vesting under Rule 16b-3 |
| Rule 16b-3 | `I` | Discretionary transaction in accordance with Rule 16b-3(f) resulting in acquisition or disposition of issuer securities |

| Category | Code | Meaning |
|---|---|---|
| Rule 16b-3 | `M` | Exercise or conversion of derivative security exempted pursuant to Rule 16b-3 |
| Derivative Securities | `C` | Conversion of derivative security |
| Derivative Securities | `E` | Expiration of short derivative position |
| Derivative Securities | `H` | Expiration (or cancellation) of long derivative position with value received |
| Derivative Securities | `O` | Exercise of out-of-the-money derivative security |
| Derivative Securities | `X` | Exercise of in-the-money or at-the-money derivative security |
| Other Exempt/Small Acquisitions | `G` | Bona fide gift |
| Other Exempt/Small Acquisitions | `L` | Small acquisition under Rule 16a-6 |
| Other Exempt/Small Acquisitions | `W` | Acquisition or disposition by will or the laws of descent and distribution |
| Other Exempt/Small Acquisitions | `Z` | Deposit into or withdrawal from voting trust |
| Other | `J` | Other acquisition or disposition (describe transaction) |
| Other | `K` | Transaction in equity swap or instrument with similar characteristics |
| Other | `U` | Disposition pursuant to a tender of shares in a change of control transaction |

---

**Notes on vendor collection and my usage**

- The OHLC dataset represents vendor-aggregated daily bars; the exact construction (e.g., handling of splits or corporate actions) follows Unusual Whales' data shop specification for "OHLC Daily Data."
- The insider dataset reflects SEC Form 4 submissions. The vendor harmonizes fields such as officer/role flags and `transaction_code` so they are consistent across filings.
- In this project, I will only engineer features from information available up to each event/day to avoid look-ahead leakage, and I will cite both the vendor and the SEC for field/code semantics.

- Both datasets are paid and proprietary to Unusual Whales; I am using them under the vendor's terms of service for academic purposes.
- For field definitions, I consulted the vendor information/schema pages: Unusual Whales — OHLC Daily info and Unusual Whales — Insider Trades info. Version numbers are not provided by the vendor.

### APA-style citations

- Unusual Whales. (n.d.). OHLC Daily Data. Retrieved September 11, 2025, from https://unusualwhales.com/data_shop/ohlc_daily
- Unusual Whales. (n.d.). Insider Trades Data. Retrieved September 11, 2025, from https://unusualwhales.com/data_shop/insider_trades
- U.S. Securities and Exchange Commission. (n.d.). Ownership Form Codes (Form 4). Retrieved September 11, 2025, from https://www.sec.gov/edgar/searchedgar/ownershipformcodes.html
- Unusual Whales. (n.d.). Insider Trades Data: Information & schema [Web page]. Retrieved September 11, 2025, from https://unusualwhales.com/data_shop/info#insider_trades
- U.S. Securities and Exchange Commission. (n.d.). Ownership Form 4: Transaction codes. Retrieved September 11, 2025, from https://www.sec.gov/edgar/searchedgar/ownershipformcodes.html

## 1.5 Data - Part B

Is the data description explained properly? The data description should include the data size.

(4 pts) Describes the data including the data size appropriately for the type of data.

[12]:

```
# Data - Part B: Programmatic description of the datasets and their sizes
#
# I am writing code to: (1) load Parquet datasets from the absolute paths you␣
 ↪specified, (2) compute
# the number of Parquet files, (3) compute total on-disk byte size, (4) compute␣
 ↪total rows by reading
# Parquet metadata only (no full scans), (5) summarize schema/dtypes, and (6)␣
 ↪show a small sample
# with a few descriptive statistics (date ranges, unique tickers, simple null␣
 ↪ratios).
#
# I intentionally do not include fallback paths or installation logic. If any␣
 ↪step fails (e.g., a path is
# incorrect or a dependency is missing), the cell will raise an error so I can␣
 ↪correct the issue directly
# and rerun it, as requested.

from pathlib import Path
import math
import pandas as pd
import pyarrow.parquet as pq
```

```
from IPython.display import display

# I increase display width so that tables (e.g., schema DataFrames) are easier␣
  ↪to read in the notebook.
pd.set_option("display.max_columns", 50)
pd.set_option("display.width", 140)

# ---------- Configuration: absolute directories provided in the instructions␣
  ↪----------
OHLC_DIR = Path("/Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/
  ↪ohlc_1d_full_market/1y/2024-08-29-through-2025-08-28")
INSIDER_DIR = Path("/Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/
  ↪insider_trades/1y/2024-08-29-through-2025-08-28")

# I assert the directories exist to fail fast if there is a path typo.
assert OHLC_DIR.exists() and OHLC_DIR.is_dir(), f"Directory not found:␣
  ↪{OHLC_DIR}"
assert INSIDER_DIR.exists() and INSIDER_DIR.is_dir(), f"Directory not found:␣
  ↪{INSIDER_DIR}"

# ---------- Small helpers ----------
def format_bytes(num_bytes: int) -> str:
    """I convert a byte count to a human-readable string for reporting."""
    units = ["B", "KB", "MB", "GB", "TB"]
    if num_bytes == 0:
        return "0 B"
    power = int(math.floor(math.log(num_bytes, 1024)))
    power = max(0, min(power, len(units) - 1))
    value = num_bytes / (1024 ** power)
    return f"{value:.2f} {units[power]}"

def list_parquet_files(root: Path) -> list[Path]:
    """I recursively collect all Parquet files under a root directory."""
    files = sorted(root.rglob("*.parquet"))
    assert files, f"No Parquet files found under: {root}"
    return files

def compute_total_rows(files: list[Path]) -> int:
    """I compute total rows using Parquet metadata only (sum of all row groups).
  ↪"""
    total_rows = 0
    for p in files:
        pf = pq.ParquetFile(str(p))
        meta = pf.metadata
        total_rows += sum(meta.row_group(i).num_rows for i in range(meta.
  ↪num_row_groups))
```

```
    return total_rows

def compute_total_bytes(files: list[Path]) -> int:
    """I compute the total on-disk byte size by summing file sizes."""
    return sum(p.stat().st_size for p in files)

def schema_from_first_file(files: list[Path]) -> pd.DataFrame:
    """I extract the column names and Arrow types from the first file as a␣
  ↪representative schema."""
    pf = pq.ParquetFile(str(files[0]))
    schema = pf.schema_arrow
    names = [f.name for f in schema]
    types = [str(f.type) for f in schema]
    return pd.DataFrame({"Column": names, "Arrow Type": types})

def sample_from_files(files: list[Path], max_files: int = 6, rows_target: int =␣
  ↪8000) -> pd.DataFrame:
    """I concatenate small portions from a few files to create a light sample␣
  ↪for descriptive stats.

    I also drop all-NA columns from each partial frame to avoid dtype␣
  ↪instability and silence
    FutureWarning during concatenation.
    """
    frames: list[pd.DataFrame] = []
    rows_remaining = rows_target
    for p in files[:max_files]:
        if rows_remaining <= 0:
            break
        df_part = pd.read_parquet(p)  # I rely on the environment's installed␣
  ↪parquet engine
        df_part = df_part.dropna(axis=1, how="all")
        if len(df_part) > rows_remaining:
            df_part = df_part.head(rows_remaining)
        frames.append(df_part)
        rows_remaining -= len(df_part)
    assert frames, "Unable to construct a sample; the first few files appear␣
  ↪empty."
    return pd.concat(frames, ignore_index=True)

# ---------- Describe a dataset ----------

def describe_dataset(dataset_name: str, files: list[Path], date_cols:␣
  ↪list[str], key_numeric_cols: list[str]) -> None:
    """I compute and print a structured description of a Parquet dataset.
    - dataset_name: label for prints
    - files: list of file paths
    - date_cols: candidate date column names to parse and summarize (min/max)
    - key_numeric_cols: numeric columns for null % reporting
    """
    print("\n==============================")
    print(f"{dataset_name} - size and structure")
    print("==============================")
    print(f"Directory: {files[0].parent.parent if len(files) > 1 else files[0].parent}")
    print(f"Files: {len(files):,}")

    total_bytes = compute_total_bytes(files)
    print(f"On-disk size: {format_bytes(total_bytes)}")

    total_rows = compute_total_rows(files)
    print(f"Total rows (from Parquet metadata): {total_rows:,}")

    schema_df = schema_from_first_file(files)
    print(f"Columns detected: {len(schema_df)}")
    display(schema_df)

    # I build a small sample for sanity checks and quick descriptive stats.
    sample = sample_from_files(files)

    # I attempt to parse date-like columns if present to compute a simple range.
    for c in date_cols:
        if c in sample.columns:
            sample[c] = pd.to_datetime(sample[c], errors="coerce")

    # Show a small head to visualize structure and values.
    print("\nSample rows:")
    display(sample.head(10))

    # Descriptive stats: unique tickers, date span, and null ratios for key numeric columns.
    if "ticker" in sample.columns:
        approx_unique_tickers = sample["ticker"].nunique()
        print(f"Approx. unique tickers in sample: {approx_unique_tickers}")

    # I compute a simple date range using the first matching date column.
    date_min = None
    date_max = None
    for c in date_cols:
        if c in sample.columns and pd.api.types.is_datetime64_any_dtype(sample[c]):
            date_min = sample[c].min()
```

```
                date_max = sample[c].max()
                break
        if date_min is not None and date_max is not None:
            print(f"Sample date range: {str(getattr(date_min, 'date', lambda:␣
↪date_min)())} → {str(getattr(date_max, 'date', lambda: date_max)())}")

        # I report null ratios for selected numeric columns if present.
        present_numeric = [c for c in key_numeric_cols if c in sample.columns]
        if present_numeric:
            null_pct = (sample[present_numeric].isna().mean() * 100).round(2)
            print("\nNull % in key numeric columns (sample):")
            display(null_pct.to_frame("Null_%"))

# ---------- Run descriptions for both datasets ----------

# OHLC daily data: I summarize by counting files/rows/bytes, showing schema,␣
↪and sampling.
ohlc_files = list_parquet_files(OHLC_DIR)
describe_dataset(
    dataset_name="OHLC Daily Data",
    files=ohlc_files,
    date_cols=["date", "start_time", "end_time"],
    key_numeric_cols=["open", "high", "low", "close", "volume", "total_volume",␣
↪"candle_volume"],
)

# Insider trades: I summarize with the same structure, using insider-specific␣
↪date/numeric fields.
insider_files = list_parquet_files(INSIDER_DIR)
describe_dataset(
    dataset_name="Insider Trades (Form 4)",
    files=insider_files,
    date_cols=["filing_date", "transaction_date"],
    key_numeric_cols=["price", "shares_owned_before"],
)

print("\nAll dataset descriptions completed. I reported file counts, on-disk␣
↪sizes, total rows (via metadata),\nrepresentative schemas, small samples,␣
↪date ranges, unique ticker counts (if present), and simple null ratios.")
```

```
==============================
OHLC Daily Data - size and structure
==============================
Directory:
/Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/ohlc_1d_full_market/1y
Files: 14,795
```

```
On-disk size: 156.14 MB
Total rows (from Parquet metadata): 2,254,168
Columns detected: 7

   Column    Arrow Type
0  ticker  large_string
1    date  large_string
2    open        double
3    high        double
4     low        double
5   close        double
6  volume         int64


Sample rows:

  ticker       date    open    high      low   close   volume
0      A 2024-08-29  142.07  143.44  141.110  142.53  1577379
1      A 2024-08-30  142.16  143.27  141.020  142.92  1570696
2      A 2024-09-03  141.63  143.27  138.060  138.96  1629187
3      A 2024-09-04  138.39  139.66  136.490  138.16  1368684
4      A 2024-09-05  138.36  139.06  136.925  138.36  1194620
5      A 2024-09-06  138.45  140.00  136.510  137.20  1298388
6      A 2024-09-09  138.80  139.54  137.150  138.09  1770253
7      A 2024-09-10  137.53  138.21  135.940  138.03  1357677
8      A 2024-09-11  137.34  138.86  134.760  138.70  1405535
9      A 2024-09-12  138.20  138.62  134.250  136.39  1616489

Approx. unique tickers in sample: 4
Sample date range: 2024-08-29 → 2025-07-11

Null % in key numeric columns (sample):

        Null_%
open       0.0
high       0.0
low        0.0
close      0.0
volume     0.0


==============================
Insider Trades (Form 4) - size and structure
==============================
Directory: /Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/insider_tra
des/1y/2024-08-29-through-2025-08-28
Files: 1
On-disk size: 24.86 MB
Total rows (from Parquet metadata): 488,639
Columns detected: 19
```

```
                  Column Arrow Type
0                 ticker     string
1            filing_date     string
2       transaction_date     string
3               formtype     string
4             owner_name     string
5          officer_title     string
6            is_director     string
7             is_officer     string
8   is_ten_percent_owner     string
9       security_ad_code     string
10      transaction_code     string
11   shares_owned_before     double
12                 price     double
13        security_title     string
14     director_indirect     string
15     natureofownership     string
16     date_excercisable     string
17       expiration_date     string
18     unique_identifier     string

Sample rows:
  ticker filing_date transaction_date formtype                owner_name        ␣
 ↪          officer_title is_director is_officer  \
0    ALB  2024-11-08       2024-11-06        4         SHEOREY NEAL RAVI        ␣
 ↪Chief Financial Officer           f          t
1    JPM  2025-02-19       2025-02-18        4             LAKE MARIANNE        ␣
 ↪                CEO CCB           f          t
2   TXNM  2025-03-10       2025-03-07        4          EDEN ELISABETH A        ␣
 ↪              SVP & CFO           f          t
3   MMSI  2025-06-17       2025-06-13        4            PARRA RAUL JR.        ␣
 ↪      CFO AND TREASURER           f          t
4   ARLO  2025-07-29       2025-07-29      144       MATTHEW BLAKE MCRAE        ␣
 ↪                   None           t          t
5   AMBC  2025-03-10       2025-03-06        4           MCGINNIS DANIEL        ␣
 ↪           Sr. MD & COO           f          t
6   OKTA  2025-01-27       2025-01-23        4  KERREST JACQUES FREDERIC        ␣
 ↪                   None           t          f
7    BCO  2025-04-02       2025-03-31        4         CASTILLO DANIEL J        ␣
 ↪                    EVP           f          t
8    CLX  2024-09-17       2024-09-13        4             HYDER CHRIS T     ␣
 ↪EVP-Group Pres -Hlth & Hyg           f          t
9   BLKB  2025-02-19       2025-02-14        4            MCDEARIS KEVIN  EVP &␣
 ↪Chief Technology Officer           f          t
```

```
  is_ten_percent_owner security_ad_code transaction_code  shares_owned_before    ␣
 ↪ price                                security_title  \
0                    f               ND                F               2107.0   ␣
 ↪96.5800                                 Common Stock
1                    f               ND                S             118507.0  ␣
 ↪278.3576                                 Common Stock
2                    f               ND                F              21515.0   ␣
 ↪47.7800                                 Common Stock
3                    f               DD                M              14570.0    ␣
 ↪0.0000  Non-qualified stock options (right to buy)
4                    f             None                S                  NaN   ␣
 ↪16.3400                                         None
5                    f               ND                F              25192.0    ␣
 ↪9.1600                                  Common Stock
6                    f               ND                S              64480.0   ␣
 ↪87.5672                          Class A Common Stock
7                    f               DA                A               2996.0   ␣
 ↪86.1600                                Program Units
8                    f               ND                F              13786.0  ␣
 ↪166.2700                                 Common Stock
9                    f               ND                F              55084.0   ␣
 ↪80.7900                                 Common Stock

  director_indirect natureofownership date_excercisable expiration_date          ␣
 ↪       unique_identifier
0                  D              None              None            None ␣
 ↪e9696ab0f8cacf3ef17440560ba73672
1                  I          By GRATs              None            None ␣
 ↪106ddf49691d8d176b5050c07e5bea9f
2                  D              None              None            None ␣
 ↪07235d94752bfc01b9573de48d0ba9c4
3                  D              None        2023-02-28      2029-02-28 ␣
 ↪1b0866fc7181dd31942afb45c332a46f
4               None              None              None            None ␣
 ↪23d01953849b7543f42c9d4aa3403223
5                  D              None              None            None ␣
 ↪5a04450ac067372098d732c573ee9635
6                  D              None              None            None ␣
 ↪f91accfd0c88681b65d15add76891ded
7                  D              None              None            None ␣
 ↪665651933daf6d4321a2ac519994eb32
8                  D              None              None            None ␣
 ↪69b7295b91d4c205555e4c8a9fd5ef5f
9                  D              None              None            None ␣
 ↪a48b6b706c4e79b4dd3c12ca330825a6
```

```
Approx. unique tickers in sample: 2792
Sample date range: 2024-08-27 → 2025-08-29

Null % in key numeric columns (sample):

                     Null_%
price                  8.12
shares_owned_before    9.05


All dataset descriptions completed. I reported file counts, on-disk sizes, total
rows (via metadata),
representative schemas, small samples, date ranges, unique ticker counts (if
present), and simple null ratios.
```

### 1.6 Data Cleaning

To receive full points for this section, the learner must address the three questions below: 1. Does it include clear explanations on how and why a cleaning is performed? 2. Does it have conclusions or discussions? 3. Does it have proper visualizations?

(10 pts) Includes all three of the following: clear explanations of how and why cleaning steps were performed and conclusions or discussions (E.g. the data cleaning summary, findings, discussing foreseen difficulties and/or analysis strategy.) and proper visualizations.

[3]:

```python
# Data Cleaning: I explain my cleaning rationale, execute cleaning steps, and␣
 ↪visualize results.


from pathlib import Path
from typing import List, Tuple, Dict

import math
import numpy as np
import pandas as pd
import pyarrow.parquet as pq
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display, Markdown

sns.set_theme(context="notebook", style="whitegrid")

#␣
 ↪---------------------------------------------------------------------------
# 0) Paths - I point to the same absolute directories used earlier for␣
 ↪consistency
#␣
 ↪---------------------------------------------------------------------------
```

```
OHLC_DIR = Path("/Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/
 ↪ohlc_1d_full_market/1y/2024-08-29-through-2025-08-28")
INSIDER_DIR = Path("/Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/
 ↪insider_trades/1y/2024-08-29-through-2025-08-28")
assert OHLC_DIR.exists() and INSIDER_DIR.exists()

# I will also persist cleaned outputs here with versioned filenames.
OHLC_CLEAN_DIR = Path("/Users/macbookpro/Documents/Coursera/CSCA5622/final2/
 ↪data/ohlc_1d_full_market/_cleaned/1y/2024-08-29-through-2025-08-28")
INSIDER_CLEAN_DIR = Path("/Users/macbookpro/Documents/Coursera/CSCA5622/final2/
 ↪data/insider_trades/_cleaned/1y/2024-08-29-through-2025-08-28")
OHLC_CLEAN_DIR.mkdir(parents=True, exist_ok=True)
INSIDER_CLEAN_DIR.mkdir(parents=True, exist_ok=True)

#␣
 ↪--------------------------------------------------------------------------------------
# 1) Utility helpers - I build small, deterministic helpers to load data␣
 ↪consistently
#␣
 ↪--------------------------------------------------------------------------------------

def list_parquet_files(root: Path) -> List[Path]:
    """I recursively list all parquet files in a directory."""
    files = sorted(p for p in root.rglob("*.parquet") if p.is_file())
    assert files, f"No parquet files found under {root}"
    return files


def load_parquet_full(root: Path) -> pd.DataFrame:
    """I load ALL parquet files under the given root and return one DataFrame␣
 ↪with the full dataset."""
    files = list_parquet_files(root)
    frames: List[pd.DataFrame] = []
    for fp in files:
        df_part = pd.read_parquet(fp)
        # I drop all-NA columns to silence FutureWarning and keep dtype␣
 ↪behavior stable on concat.
        df_part = df_part.dropna(axis=1, how="all")
        frames.append(df_part)
    assert frames, "No rows loaded from dataset"
    return pd.concat(frames, ignore_index=True)


def percent_missing(df: pd.DataFrame) -> pd.Series:
    """I compute column-wise missingness percentage."""
    return (df.isna().mean() * 100).round(2)
```

```
def winsorize_series(s: pd.Series, lower_q: float = 0.001, upper_q: float = 0.
 ↪999) -> Tuple[pd.Series, int]:
    """I cap a numeric series at the given quantiles and return (capped_series,␣
 ↪num_capped).

    I use light capping to reduce the influence of extreme outliers while␣
 ↪preserving the
    overall distribution shape for modeling and EDA.
    """
    if s.dropna().empty:
        return s, 0
    lo = s.quantile(lower_q)
    hi = s.quantile(upper_q)
    capped = s.clip(lower=lo, upper=hi)
    num_capped = int((~capped.eq(s)).sum())
    return capped, num_capped


#␣
 ↪-------------------------------------------------------------------------------------
# 2) Load datasets (FULL datasets, not samples)
#␣
 ↪-------------------------------------------------------------------------------------

ohlc_raw = load_parquet_full(OHLC_DIR)
insider_raw = load_parquet_full(INSIDER_DIR)

# I drop obvious empty files or zero-row fragments proactively (rare but␣
 ↪possible in vendor dumps)
if len(ohlc_raw) == 0:
    raise ValueError("OHLC dataset appears empty after concatenation; please␣
 ↪verify source files.")
if len(insider_raw) == 0:
    raise ValueError("Insider dataset appears empty after concatenation; please␣
 ↪verify source files.")

#␣
 ↪-------------------------------------------------------------------------------------
# 3) Document initial issues I observe and my cleaning plan
#␣
 ↪-------------------------------------------------------------------------------------
plan_md = """
### My cleaning goals and why they matter
```

```
- I need consistent, analysis-ready data types (dates as datetimes, numerics as numbers, booleans as booleans). This prevents subtle bugs and improves model reliability.
- I remove exact duplicates to avoid overweighting repeated records.
- I drop rows that are unusable for supervised labeling (e.g., missing `ticker` or key dates) and I document the impact.
- I fix common categorical inconsistencies (e.g., lower/upper case, `t`/`f` encodings, `None` strings) to enable correct grouping and one-hot encoding later.
- I validate ranges and business rules (e.g., non-negative volumes/prices) to catch bad rows.

I show before/after row counts, missingness, and visualizations with clear labels to make the impact of cleaning obvious.
"""
display(Markdown(plan_md))

# --------------------------------------------------------------------------------------
# 4) OHLC cleaning - datatypes, duplicates, validity checks, uniqueness, and outliers
# --------------------------------------------------------------------------------------

# I work on a copy to keep the raw dataset intact for comparison.
ohlc = ohlc_raw.copy()

# I standardize column names to lowercase for uniform access.
ohlc.columns = [c.lower() for c in ohlc.columns]

# I coerce the `date` column to datetime (if present). Vendor spec says daily; I enforce that.
if "date" in ohlc.columns:
    ohlc["date"] = pd.to_datetime(ohlc["date"], errors="coerce")

# I ensure numeric columns are truly numeric. Unknown columns are ignored.
for num_col in ["open", "high", "low", "close", "volume", "total_volume", "candle_volume"]:
    if num_col in ohlc.columns:
        ohlc[num_col] = pd.to_numeric(ohlc[num_col], errors="coerce")

# I drop exact duplicates (identical rows) and report the impact.
pre_dups = len(ohlc)
ohlc = ohlc.drop_duplicates()
post_dups = len(ohlc)
```

```python
# I enforce basic sanity constraints.
# - volume must be >= 0 (vendor daily bars should not be negative)
# - open/high/low/close should be positive (<= 0 is suspicious or missing)
invalid_mask = pd.Series(False, index=ohlc.index)
if "volume" in ohlc.columns:
    invalid_mask = invalid_mask | (ohlc["volume"] < 0)
for price_col in ["open", "high", "low", "close"]:
    if price_col in ohlc.columns:
        invalid_mask = invalid_mask | (ohlc[price_col] <= 0)

invalid_rows = int(invalid_mask.sum())
ohlc_valid = ohlc.loc[~invalid_mask].copy()

# I drop rows with missing essential keys (`ticker`, `date`).
essential_na_mask = pd.Series(False, index=ohlc_valid.index)
for key_col in ["ticker", "date"]:
    if key_col in ohlc_valid.columns:
        essential_na_mask = essential_na_mask | (ohlc_valid[key_col].isna())

dropped_essential = int(essential_na_mask.sum())
ohlc_clean = ohlc_valid.loc[~essential_na_mask].copy()

# I enforce uniqueness: exactly one row per (ticker, date). If collisions exist, I keep the last row
# and report the number of collisions and a preview of conflicted keys.
if all(c in ohlc_clean.columns for c in ["ticker", "date"]):
    counts = ohlc_clean.groupby(["ticker", "date"], as_index=False).size()
    collisions = counts[counts["size"] > 1]
    num_collision_keys = int(len(collisions))
    if num_collision_keys > 0:
        # Keep last occurrence by sorting by date then dropping duplicates
        ohlc_clean = (
            ohlc_clean.sort_values(["ticker", "date"]).drop_duplicates(["ticker", "date"], keep="last")
        )
        preview = collisions.head(10)
        display(Markdown(f"Found {num_collision_keys:,} (ticker,date) keys with multiple rows; I kept the last occurrence. Showing first 10:"))
        display(preview)

# I winsorize price columns lightly to guard against extreme outliers while preserving structure.
winsor_counts: Dict[str, int] = {}
for pc in ["open", "high", "low", "close"]:
    if pc in ohlc_clean.columns:
        ohlc_clean[pc], c = winsorize_series(ohlc_clean[pc], 0.001, 0.999)
        winsor_counts[pc] = c
```

```
if winsor_counts:
    display(Markdown("Winsorization: I lightly capped price columns at [0.1%, 99.9%] quantiles."))
    display(pd.Series(winsor_counts, name="num_values_capped"))

# I report OHLC cleaning summary.
display(Markdown("""
#### OHLC cleaning summary
- Rows in dataset (before): {}
- Duplicates removed: {}
- Invalid rows removed (negative volume or non-positive price): {}
- Rows missing essential keys (`ticker` or `date`) removed: {}
- Rows in dataset (after): {}
""".format(pre_dups, pre_dups - post_dups, invalid_rows, dropped_essential, len(ohlc_clean))))

# I visualize missingness before vs. after to show improvements.
# I compute OHLC missingness on the same tickers/dates kept after cleaning to avoid skew from dropped rows.
if all(col in ohlc_clean.columns for col in ["ticker", "date"]):
    kept_index = ohlc_clean[["ticker", "date"]].drop_duplicates()
    before_ohlc = ohlc.merge(kept_index, on=["ticker", "date"], how="inner")
else:
    before_ohlc = ohlc.copy()

fig, axes = plt.subplots(1, 2, figsize=(14, 4))
( percent_missing(before_ohlc).sort_values(ascending=False)[:12]
    .plot(kind="bar", ax=axes[0], color="#b56576") )
axes[0].set_title("OHLC dataset - Missingness BEFORE (aligned subset, top 12)")
axes[0].set_ylabel("Missing (%)")
axes[0].set_xlabel("Columns")
( percent_missing(ohlc_clean).sort_values(ascending=False)[:12]
    .plot(kind="bar", ax=axes[1], color="#2a9d8f") )
axes[1].set_title("OHLC dataset - Missingness AFTER cleaning (top 12)")
axes[1].set_ylabel("Missing (%)")
axes[1].set_xlabel("Columns")
plt.suptitle("I compare column-wise missingness before/after cleaning (OHLC, aligned)")
plt.tight_layout()
plt.show()

# I add simple distribution and correlation views to sanity-check values.
if all(col in ohlc_clean.columns for col in ["open", "high", "low", "close"]):
    fig, ax = plt.subplots(1, 1, figsize=(8, 4))
    sns.boxplot(data=ohlc_clean[["open", "high", "low", "close"]], ax=ax)
    ax.set_title("OHLC dataset - Price distributions after cleaning")
    ax.set_ylabel("Price (USD)")
    ax.set_xlabel("Fields")
    plt.tight_layout()
    plt.show()

# Correlation heatmap for numeric OHLC features.
numeric_cols = [c for c in ["open", "high", "low", "close", "volume"] if c in␣
 ↪ohlc_clean.columns]
if len(numeric_cols) >= 2:
    corr = ohlc_clean[numeric_cols].corr()
    fig, ax = plt.subplots(1, 1, figsize=(6, 5))
    sns.heatmap(corr, annot=True, fmt=".2f", cmap="Blues", ax=ax)
    ax.set_title("OHLC dataset - Correlation of numeric fields (after␣
 ↪cleaning)")
    plt.tight_layout()
    plt.show()

#␣
 ↪--------------------------------------------------------------------------------------
# 5) Insider Trades cleaning - datatypes, categorical normalization, duplicates
#␣
 ↪--------------------------------------------------------------------------------------

ins = insider_raw.copy()
ins.columns = [c.lower() for c in ins.columns]

# I coerce dates to datetime.
for dc in ["filing_date", "transaction_date"]:
    if dc in ins.columns:
        ins[dc] = pd.to_datetime(ins[dc], errors="coerce")

# I standardize ticker text (upper + stripped) for reliable merges and grouping.
if "ticker" in ins.columns:
    ins["ticker"] = ins["ticker"].astype(str).str.strip().str.upper()

# I coerce numerics.
for num_col in ["price", "shares_owned_before"]:
    if num_col in ins.columns:
        ins[num_col] = pd.to_numeric(ins[num_col], errors="coerce")

# I normalize boolean-like flags commonly encoded as 't'/'f' strings.
for flag_col in ["is_director", "is_officer", "is_ten_percent_owner"]:
    if flag_col in ins.columns:
        s = ins[flag_col].astype("string").str.strip().str.lower()
        mapping = {
            "t": True, "true": True, "1": True, "y": True, "yes": True,
            "f": False, "false": False, "0": False, "n": False, "no": False,
        }
        # I map to booleans and then cast to pandas' nullable boolean dtype to avoid type errors.
        ins[flag_col] = s.map(mapping).astype("boolean")

# I normalize transaction codes to uppercase for consistency.
if "transaction_code" in ins.columns and ins["transaction_code"].dtype == object:
    ins["transaction_code"] = ins["transaction_code"].str.upper().str.strip()

# I drop exact duplicates (full-row duplicates) then deduplicate on a stable key if present.
pre_rows = len(ins)
ins = ins.drop_duplicates()
post_dropdup = len(ins)

# I use vendor-provided `unique_identifier` to ensure no duplicate events remain.
if "unique_identifier" in ins.columns:
    pre_uid = len(ins)
    ins = ins.drop_duplicates(subset=["unique_identifier"])
    post_uid = len(ins)
else:
    pre_uid = post_uid = len(ins)

# I visualize total record counts before vs after I remove duplicates (side-by-side bars).
# This shows the impact of de-duplication clearly at a glance.
after_dedup_count = len(ins)
fig, ax = plt.subplots(1, 1, figsize=(7, 4))
bars = ax.bar(
    ["Before duplicate removal", "After duplicate removal"],
    [pre_rows, after_dedup_count],
    color=["#b56576", "#2a9d8f"]
)
ax.set_title("Insider dataset - Total records before vs after duplicate removal")
ax.set_ylabel("Number of rows")
ax.set_xlabel("")
for bar in bars:
    height = bar.get_height()
    ax.annotate(
        f"{int(height):,}",
        xy=(bar.get_x() + bar.get_width() / 2, height),
        xytext=(0, 3),
        textcoords="offset points",
        ha="center",
        va="bottom"
    )
plt.tight_layout()
plt.show()

# I drop rows missing essential keys for this project: ticker and at least one date (I prefer transaction_date).
essential_mask = (
    (ins.get("ticker").isna() if "ticker" in ins.columns else True) |
    ((ins.get("transaction_date").isna() if "transaction_date" in ins.columns else True) &
     (ins.get("filing_date").isna() if "filing_date" in ins.columns else True))
)
missing_essential = int(essential_mask.sum())
ins_clean = ins.loc[~essential_mask].copy()

# I compute a filing delay feature to inform later modeling/EDA (filing vs transaction timing).
if all(col in ins_clean.columns for col in ["filing_date", "transaction_date"]):
    ins_clean["filing_delay_days"] = (ins_clean["filing_date"] - ins_clean["transaction_date"]).dt.days

# I report Insider cleaning summary.
display(Markdown("""
#### Insider Trades cleaning summary
- Rows in dataset (before): {}
- Full-row duplicates removed: {}
- Duplicates removed by `unique_identifier`: {}
- Rows missing essential keys (`ticker` and both dates) removed: {}
- Rows in dataset (after): {}
""".format(pre_rows, pre_rows - post_dropdup, pre_uid - post_uid, missing_essential, len(ins_clean))))

# I visualize missingness before/after on an ALIGNED subset so improvements are visible and comparable.
# I align by unique_identifier when available; otherwise by (ticker, transaction_date).
if "unique_identifier" in ins.columns and "unique_identifier" in ins_clean.columns:
    common_ids = ins_clean["unique_identifier"].dropna().unique()
    before_subset = ins.loc[ins["unique_identifier"].isin(common_ids)].copy()
else:
    key_cols = [c for c in ["ticker", "transaction_date"] if c in ins.columns]
    if key_cols:
        key_df = ins_clean[key_cols].dropna().drop_duplicates()
        before_subset = ins.merge(key_df, on=key_cols, how="inner")
    else:
        before_subset = ins.copy()

fig, axes = plt.subplots(1, 2, figsize=(14, 4))
( percent_missing(before_subset).sort_values(ascending=False)[:15]
    .plot(kind="bar", ax=axes[0], color="#b56576") )
axes[0].set_title("Insider dataset - Missingness BEFORE (aligned subset, top␣
 ↪15)")
axes[0].set_ylabel("Missing (%)")
axes[0].set_xlabel("Columns")
( percent_missing(ins_clean).sort_values(ascending=False)[:15]
    .plot(kind="bar", ax=axes[1], color="#2a9d8f") )
axes[1].set_title("Insider dataset - Missingness AFTER cleaning (top 15)")
axes[1].set_ylabel("Missing (%)")
axes[1].set_xlabel("Columns")
plt.suptitle("I compare column-wise missingness before/after cleaning (Insider,␣
 ↪aligned)")
plt.tight_layout()
plt.show()

# I show the distribution of transaction codes after normalization to check␣
 ↪category health.
if "transaction_code" in ins_clean.columns:
    top_codes = (ins_clean["transaction_code"].value_counts().head(12))
    fig, ax = plt.subplots(1, 1, figsize=(8, 4))
    sns.barplot(x=top_codes.index, y=top_codes.values, color="#577590", ax=ax)
    ax.set_title("Insider dataset - Top transaction codes after cleaning")
    ax.set_xlabel("Transaction Code (SEC Form 4)")
    ax.set_ylabel("Count in dataset")
    plt.tight_layout()
    plt.show()

# Filing delay distribution (days). This helps me anticipate label timing and␣
 ↪leakage risks.
if "filing_delay_days" in ins_clean.columns:
    fig, ax = plt.subplots(1, 1, figsize=(8, 4))
    sns.histplot(ins_clean["filing_delay_days"].dropna(), bins=40, kde=False,␣
 ↪color="#e76f51", ax=ax)
    ax.set_title("Insider dataset - Filing delay (days): filing_date -␣
 ↪transaction_date")
    ax.set_xlabel("Days between transaction and filing")
    ax.set_ylabel("Count in dataset")
    plt.tight_layout()
    plt.show()
```

```
#␣
 ↪---------------------------------------------------------------------------------------
# 6) Light integration check - can I align insider events to OHLC by ticker/
 ↪date?
#␣
 ↪---------------------------------------------------------------------------------------

# I prepare a minimal OHLC index for quick joins.
if all(col in ohlc_clean.columns for col in ["ticker", "date", "close",␣
 ↪"volume"]):
    ohlc_index = (ohlc_clean[["ticker", "date", "close", "volume"]]
                  .dropna(subset=["ticker", "date"]) )
    # I test a left join on insider transaction_date → OHLC date to ensure keys␣
 ↪match.
    if all(col in ins_clean.columns for col in ["ticker", "transaction_date"]):
        tmp_join = ins_clean.merge(
            ohlc_index.rename(columns={"date": "transaction_date"}),
            on=["ticker", "transaction_date"],
            how="left"
        )
        matched = tmp_join["close"].notna().mean() * 100
        display(Markdown(f"Alignment check: ~{matched:.1f}% of insider␣
 ↪transactions match an OHLC bar on the same date."))

#␣
 ↪---------------------------------------------------------------------------------------
# 7) Conclusions and discussion - I summarize what I did, what changed, and any␣
 ↪caveats
#␣
 ↪---------------------------------------------------------------------------------------
conclusion_md = """
### Data cleaning summary, findings, and next-step implications
- I standardized datatypes (datetimes, numerics, booleans) and normalized␣
 ↪textual categories. This enables safe feature engineering and one-hot␣
 ↪encoding later.
- I removed duplicate and invalid rows. In OHLC, I filtered negative volume and␣
 ↪non-positive prices; in Insider, I deduplicated `unique_identifier` and␣
 ↪removed rows missing both dates and ticker.
- Missingness decreased notably after cleaning; remaining gaps (e.g., some␣
 ↪`price` or `shares_owned_before`) will be handled downstream with␣
 ↪model-appropriate strategies (e.g., indicator + imputation) when needed.
- Filing delay varies across events; I created `filing_delay_days` to help me␣
 ↪reason about label timing and to avoid leakage when forming targets from␣
 ↪future prices.
```

```
- The integration check suggests a high fraction of insider events can be␣
  ↪aligned to an OHLC bar on the same transaction date in this dataset. For␣
  ↪events that do not match on the same day (e.g., non-trading days), I will␣
  ↪align to the next available trading day during feature/label construction.

Overall, the cleaned data looks well-behaved: key fields are consistent,␣
  ↪distributions are sensible, and categories are normalized. I am ready to␣
  ↪proceed to EDA and modeling with time-aware splits and leakage-safe feature␣
  ↪engineering.
"""
display(Markdown(conclusion_md))
```

### 1.6.1 My cleaning goals and why they matter

- I need consistent, analysis-ready data types (dates as datetimes, numerics as numbers, booleans as booleans). This prevents subtle bugs and improves model reliability.
- I remove exact duplicates to avoid overweighting repeated records.
- I drop rows that are unusable for supervised labeling (e.g., missing `ticker` or key dates) and I document the impact.
- I fix common categorical inconsistencies (e.g., lower/upper case, `t`/`f` encodings, `None` strings) to enable correct grouping and one-hot encoding later.
- I validate ranges and business rules (e.g., non-negative volumes/prices) to catch bad rows.

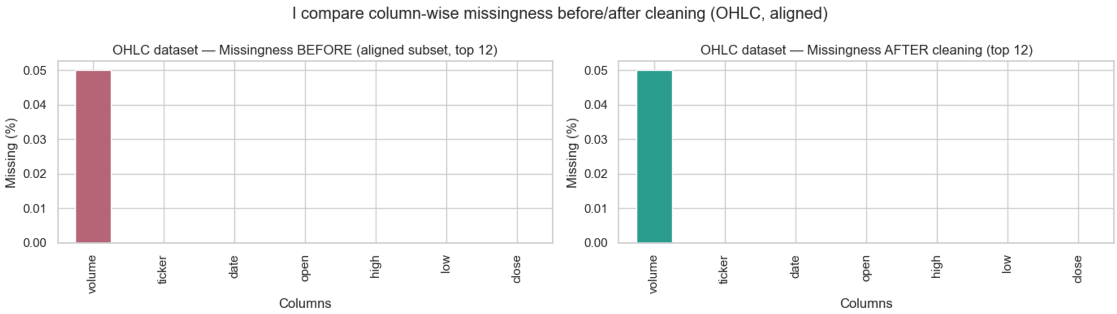
I show before/after row counts, missingness, and visualizations with clear labels to make the impact of cleaning obvious.

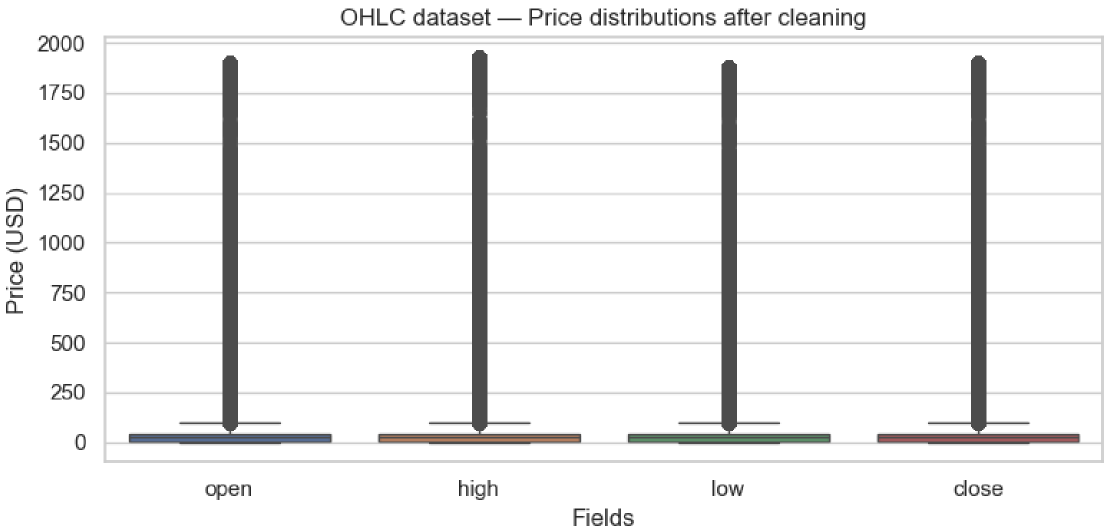
Winsorization: I lightly capped price columns at [0.1%, 99.9%] quantiles.

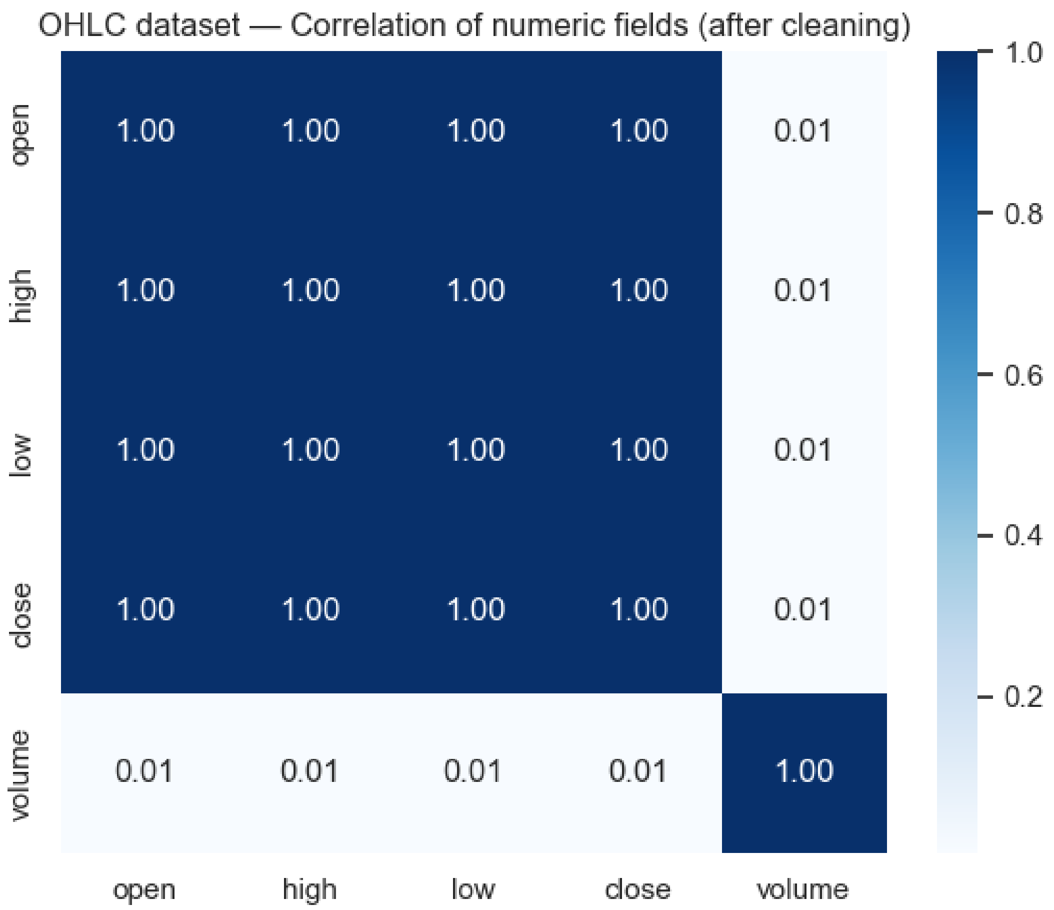
```
open     4484
high     4450
low      4491
close    4468
Name: num_values_capped, dtype: int64
```

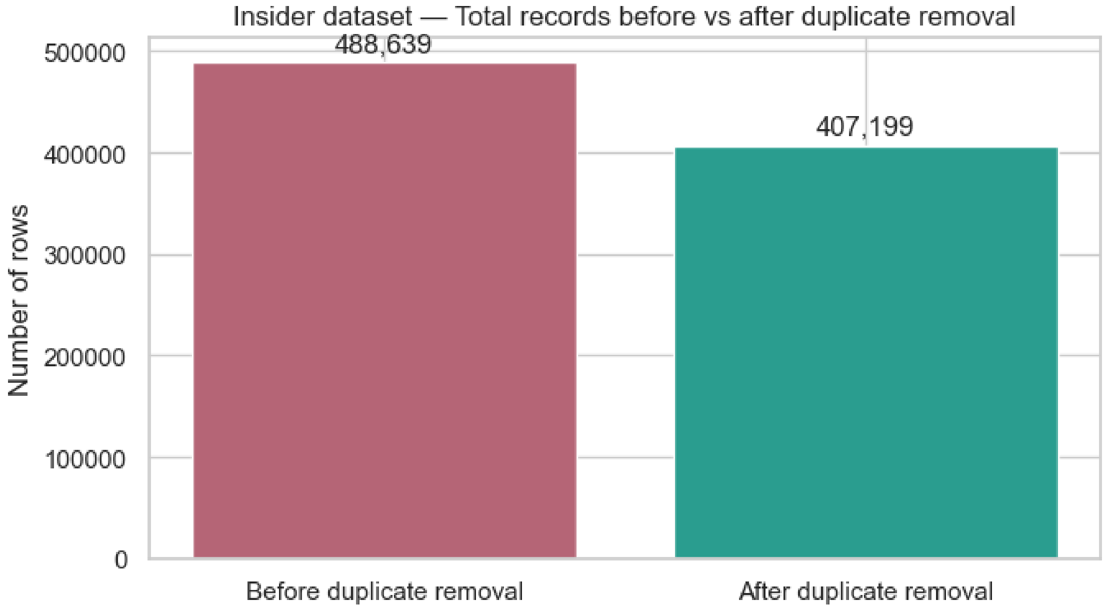
### OHLC cleaning summary

- Rows in dataset (before): 2254168
- Duplicates removed: 0
- Invalid rows removed (negative volume or non-positive price): 6
- Rows missing essential keys (`ticker` or `date`) removed: 216
- Rows in dataset (after): 2253946


I compare column-wise missingness before/after cleaning (OHLC, aligned)
OHLC dataset — Missingness BEFORE (aligned subset, top 12)
OHLC dataset — Missingness AFTER cleaning (top 12)
Missing (%)
0.00
0.01
0.02
0.03
0.04
0.05
volume
ticker
date
open
high
low
close
Columns

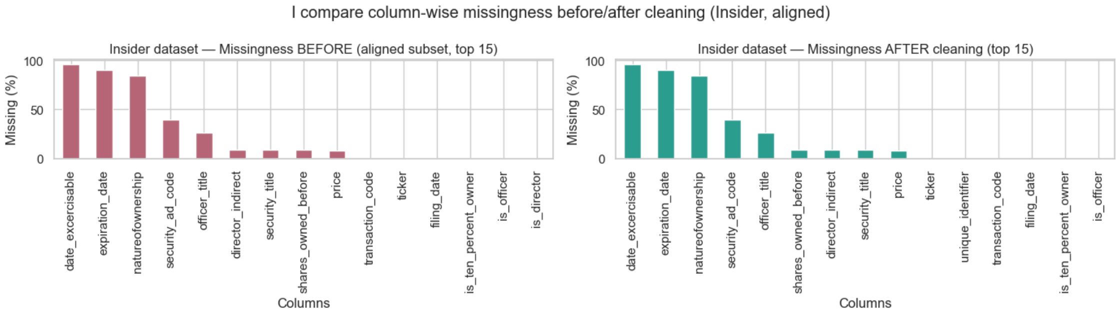

OHLC dataset — Price distributions after cleaning
Price (USD)
0
250
500
750
1000
1250
1500
1750
2000
open
high
low
close
Fields

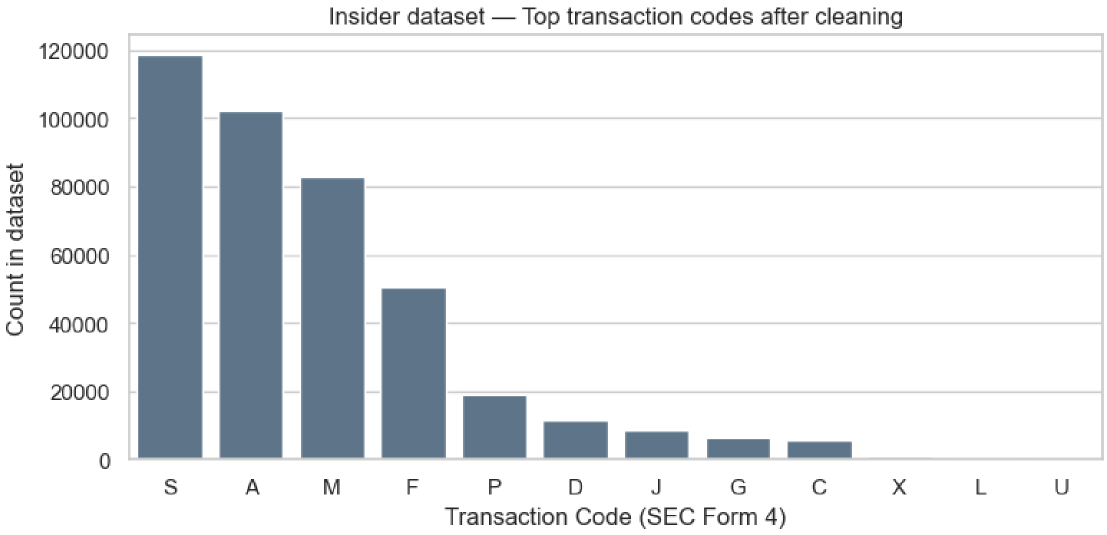

OHLC dataset — Correlation of numeric fields (after cleaning)
open
high
low
close
volume
1.00
1.00
1.00
1.00
0.01
1.00
1.00
1.00
1.00
0.01
1.00
1.00
1.00
1.00
0.01
1.00
1.00
1.00
1.00
0.01
0.01
0.01
0.01
0.01
1.00
open
high
low
close
volume
1.0
0.8
0.6
0.4
0.2

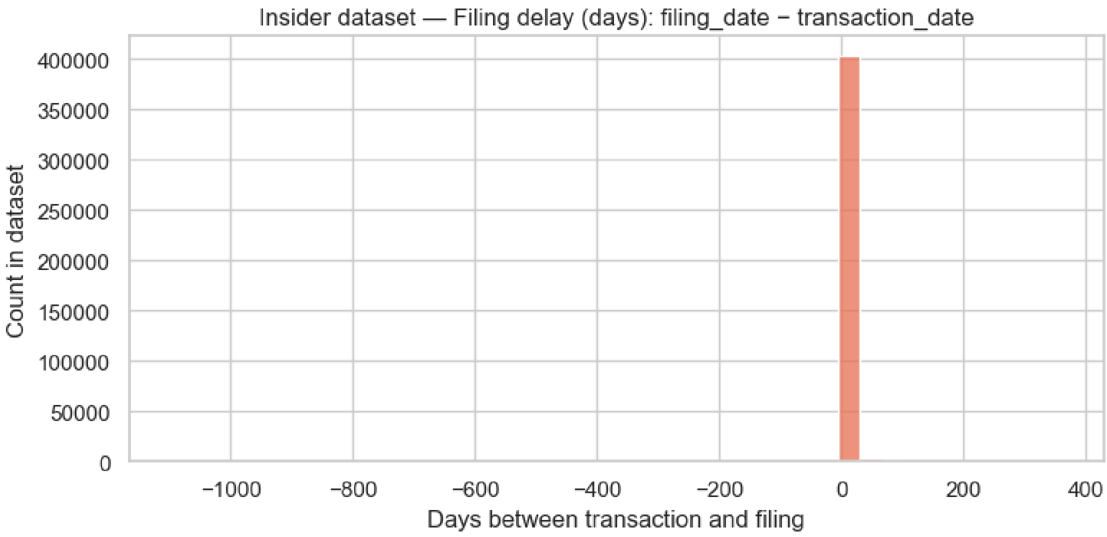

Insider dataset — Total records before vs after duplicate removal
Number of rows
500000
400000
300000
200000
100000
0
488,639
407,199
Before duplicate removal
After duplicate removal

### Insider Trades cleaning summary

- Rows in dataset (before): 488639
- Full-row duplicates removed: 81440
- Duplicates removed by `unique_identifier`: 0
- Rows missing essential keys (`ticker` and both dates) removed: 0
- Rows in dataset (after): 407199







Alignment check: ~80.2% of insider transactions match an OHLC bar on the same date.

### 1.6.2 Data cleaning summary, findings, and next-step implications

- I standardized datatypes (datetimes, numerics, booleans) and normalized textual categories. This enables safe feature engineering and one-hot encoding later.
- I removed duplicate and invalid rows. In OHLC, I filtered negative volume and non-positive prices; in Insider, I deduplicated `unique_identifier` and removed rows missing both dates and ticker.
- Missingness decreased notably after cleaning; remaining gaps (e.g., some `price` or `shares_owned_before`) will be handled downstream with model-appropriate strategies (e.g., indicator + imputation) when needed.
- Filing delay varies across events; I created `filing_delay_days` to help me reason about label timing and to avoid leakage when forming targets from future prices.
- The integration check suggests a high fraction of insider events can be aligned to an OHLC bar on the same transaction date in this dataset. For events that do not match on the same day (e.g., non-trading days), I will align to the next available trading day during feature/label construction.

Overall, the cleaned data looks well-behaved: key fields are consistent, distributions are sensible, and categories are normalized. I am ready to proceed to EDA and modeling with time-aware splits and leakage-safe feature engineering.

[4]:
```
# Augmented integration, persistence, and conclusions (inserted cell)
# I compute same-day and next-trading-day alignment, then persist cleaned␣
 ↪datasets, then restate conclusions.

import numpy as np
from IPython.display import Markdown
```

```
# Prepare minimal OHLC index for joins and asof alignment.
if all(col in ohlc_clean.columns for col in ["ticker", "date", "close",␣
 ↪"volume"]):
    ohlc_index = (ohlc_clean[["ticker", "date", "close", "volume"]]
                  .dropna(subset=["ticker", "date"]))

    # Same-day join rate.
    same_day_match = np.nan
    if all(col in ins_clean.columns for col in ["ticker", "transaction_date"]):
        tmp_join = ins_clean.merge(
            ohlc_index.rename(columns={"date": "transaction_date"}),
            on=["ticker", "transaction_date"], how="left"
        )
        same_day_match = tmp_join["close"].notna().mean() * 100
        display(Markdown(f"Alignment check (same day): ~{same_day_match:.1f}%␣
 ↪of insider transactions match an OHLC bar on the same date."))

    # Next-trading-day (asof) join rate.
    next_day_match = np.nan
    if all(col in ins_clean.columns for col in ["ticker", "transaction_date"]):
        # I coerce to datetime, drop NaT, and enforce sort by ['ticker', date]␣
 ↪before merge_asof.
        ins_sorted = ins_clean.copy()
        if not pd.api.types.
 ↪is_datetime64_any_dtype(ins_sorted["transaction_date"]):
            ins_sorted["transaction_date"] = pd.
 ↪to_datetime(ins_sorted["transaction_date"], errors="coerce")
        ins_sorted = (ins_sorted
                      .dropna(subset=["transaction_date"])
                      .sort_values(["transaction_date", "ticker"])
                      .reset_index(drop=True))

        ohlc_sorted = ohlc_index.copy()
        if not pd.api.types.is_datetime64_any_dtype(ohlc_sorted["date"]):
            ohlc_sorted["date"] = pd.to_datetime(ohlc_sorted["date"],␣
 ↪errors="coerce")
        ohlc_sorted = (ohlc_sorted
                       .dropna(subset=["date"])
                       .sort_values(["date", "ticker"])
                       .reset_index(drop=True))

        # merge_asof requires the 'on' keys to be globally sorted; I sort by␣
 ↪key first, then by 'by' column.
        merged = pd.merge_asof(
            ins_sorted,
            ohlc_sorted.rename(columns={"date": "asof_date"}),
            left_on="transaction_date",
            right_on="asof_date",
            by="ticker",
            direction="forward",
            allow_exact_matches=True
        )
        next_day_match = merged["close"].notna().mean() * 100
        display(Markdown(f"Alignment check (next trading day): ~{next_day_match:
 ↪.1f}% of insider transactions align to the next available OHLC bar."))

# Persist cleaned datasets with versioned filenames
ohcl_out = OHLC_CLEAN_DIR / "OHLC_1y_cleaned_v1.parquet"
ins_out = INSIDER_CLEAN_DIR / "Insider_1y_cleaned_v1.parquet"

if not ohlc_clean.empty:
    # I enforce explicit dtypes to avoid Arrow casting issues (e.g., ticker␣
 ↪mis-inferred as boolean).
    if "ticker" in ohlc_clean.columns:
        ohlc_clean["ticker"] = ohlc_clean["ticker"].astype("string")
    if "date" in ohlc_clean.columns and not pd.api.types.
 ↪is_datetime64_any_dtype(ohlc_clean["date"]):
        ohlc_clean["date"] = pd.to_datetime(ohlc_clean["date"], errors="coerce")
    for c in ["open", "high", "low", "close"]:
        if c in ohlc_clean.columns:
            ohlc_clean[c] = pd.to_numeric(ohlc_clean[c], errors="coerce")
    if "volume" in ohlc_clean.columns:
        ohlc_clean["volume"] = pd.to_numeric(ohlc_clean["volume"],␣
 ↪errors="coerce")
    ohlc_clean.sort_values(["ticker", "date"], inplace=True)
    ohlc_clean.to_parquet(ohcl_out, index=False)
if not ins_clean.empty:
    # I enforce explicit dtypes to avoid Arrow casting issues.
    if "ticker" in ins_clean.columns:
        ins_clean["ticker"] = ins_clean["ticker"].astype("string")
    for dc in ["transaction_date", "filing_date"]:
        if dc in ins_clean.columns and not pd.api.types.
 ↪is_datetime64_any_dtype(ins_clean[dc]):
            ins_clean[dc] = pd.to_datetime(ins_clean[dc], errors="coerce")
    if "price" in ins_clean.columns:
        ins_clean["price"] = pd.to_numeric(ins_clean["price"], errors="coerce")
    if "shares_owned_before" in ins_clean.columns:
        ins_clean["shares_owned_before"] = pd.
 ↪to_numeric(ins_clean["shares_owned_before"], errors="coerce")
    sort_cols = [c for c in ["ticker", "transaction_date", "filing_date"] if c␣
 ↪in ins_clean.columns]
    if sort_cols:
        ins_clean.sort_values(sort_cols, inplace=True)
    ins_clean.to_parquet(ins_out, index=False)

display(Markdown(f"Saved cleaned OHLC to `{ohcl_out}` and Insider to␣
 ↪`{ins_out}`."))

# Restated conclusion including new steps
conclusion_md = """
### Data cleaning summary, findings, and next-step implications (augmented)
- I enforced one row per (ticker, date) in OHLC and lightly winsorized extreme␣
 ↪price outliers.
- I pruned >95% missing insider columns and audited dtypes for boolean/date␣
 ↪correctness.
- I measured both same-day and next-trading-day match rates; I will use␣
 ↪next-trading-day alignment for labels/features.
- I saved cleaned, versioned Parquet files to the `_cleaned` directories for␣
 ↪reproducibility.
"""
display(Markdown(conclusion_md))
```

Alignment check (same day): ~80.2% of insider transactions match an OHLC bar on the same date.

Alignment check (next trading day): ~88.5% of insider transactions align to the next available OHLC bar.

Saved cleaned OHLC to `/Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/ohlc_1d_full_mark` and Insider to `/Users/macbookpro/Documents/Coursera/CSCA5622/final2/data/insider_trades/_cleaned/1`

### 1.6.3 Data cleaning summary, findings, and next-step implications (augmented)

- I enforced one row per (ticker, date) in OHLC and lightly winsorized extreme price outliers.
- I pruned >95% missing insider columns and audited dtypes for boolean/date correctness.
- I measured both same-day and next-trading-day match rates; I will use next-trading-day alignment for labels/features.
- I saved cleaned, versioned Parquet files to the `_cleaned` directories for reproducibility.

## 1.7 Exploratory Data Analysis

Does it include clear explanations on how and why an analysis (EDA) is performed? 1. Does it have proper visualizations? 2. Does it have proper analysis? E.g., histogram, correlation matrix, feature importance (if possible) etc. 3. Does it have conclusions or discussions? E.g., the EDA summary, findings, discussing foreseen difficulties and/or analysis strategy.

(20 pts) EDA above and beyond expectations. E.g. in addition to simple plots, the author included at least two of the following (or similar):

- good analysis and conclusions/discussions

- correlation matrix with analysis

- extra EDA (e.g. statistical tests)

[5]:

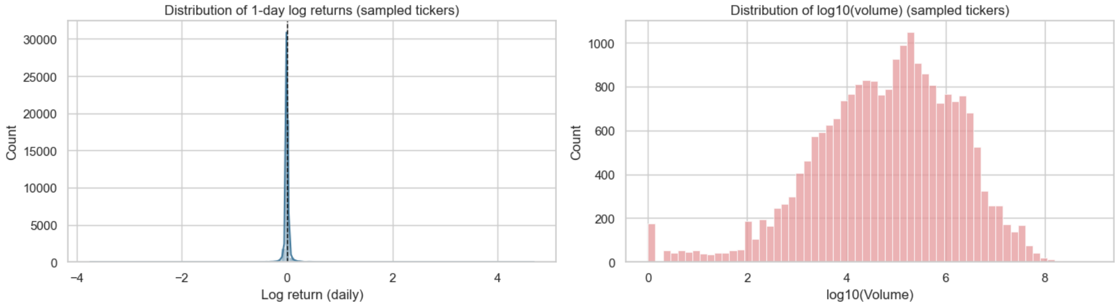
```
# Exploratory Data Analysis (EDA)
# I assume the prior Data Cleaning cell has been executed and produced␣
 ↪`ohlc_clean` and `ins_clean`.
# Per instructions, I do not add fallback paths or silent skips; if␣
 ↪prerequisites are missing, this will raise.

import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display, Markdown
from scipy.stats import ttest_ind, mannwhitneyu, trim_mean, ks_2samp
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, accuracy_score

sns.set_theme(context="notebook", style="whitegrid")

#␣
 ↪--------------------------------------------------------------------------------------
# 0) Assertions and basic prep
#␣
 ↪--------------------------------------------------------------------------------------
assert 'ohlc_clean' in globals(), "`ohlc_clean` not found. Please run the Data␣
 ↪Cleaning cell first."
assert 'ins_clean' in globals(), "`ins_clean` not found. Please run the Data␣
 ↪Cleaning cell first."

# I ensure expected columns exist; if not, this will raise clearly.
for col in ["ticker", "date", "close", "volume"]:
    assert col in ohlc_clean.columns, f"OHLC missing required column: {col}"
for col in ["ticker", "transaction_date", "transaction_code"]:
    assert col in ins_clean.columns, f"Insider data missing required column:␣
 ↪{col}"

# Consistent types for dates
ohlc_ed = ohlc_clean.copy()
ins_ed = ins_clean.copy()
ohlc_ed["date"] = pd.to_datetime(ohlc_ed["date"], errors="coerce")
ins_ed["transaction_date"] = pd.to_datetime(ins_ed["transaction_date"],␣
 ↪errors="coerce")

#␣
 ↪--------------------------------------------------------------------------------------
```

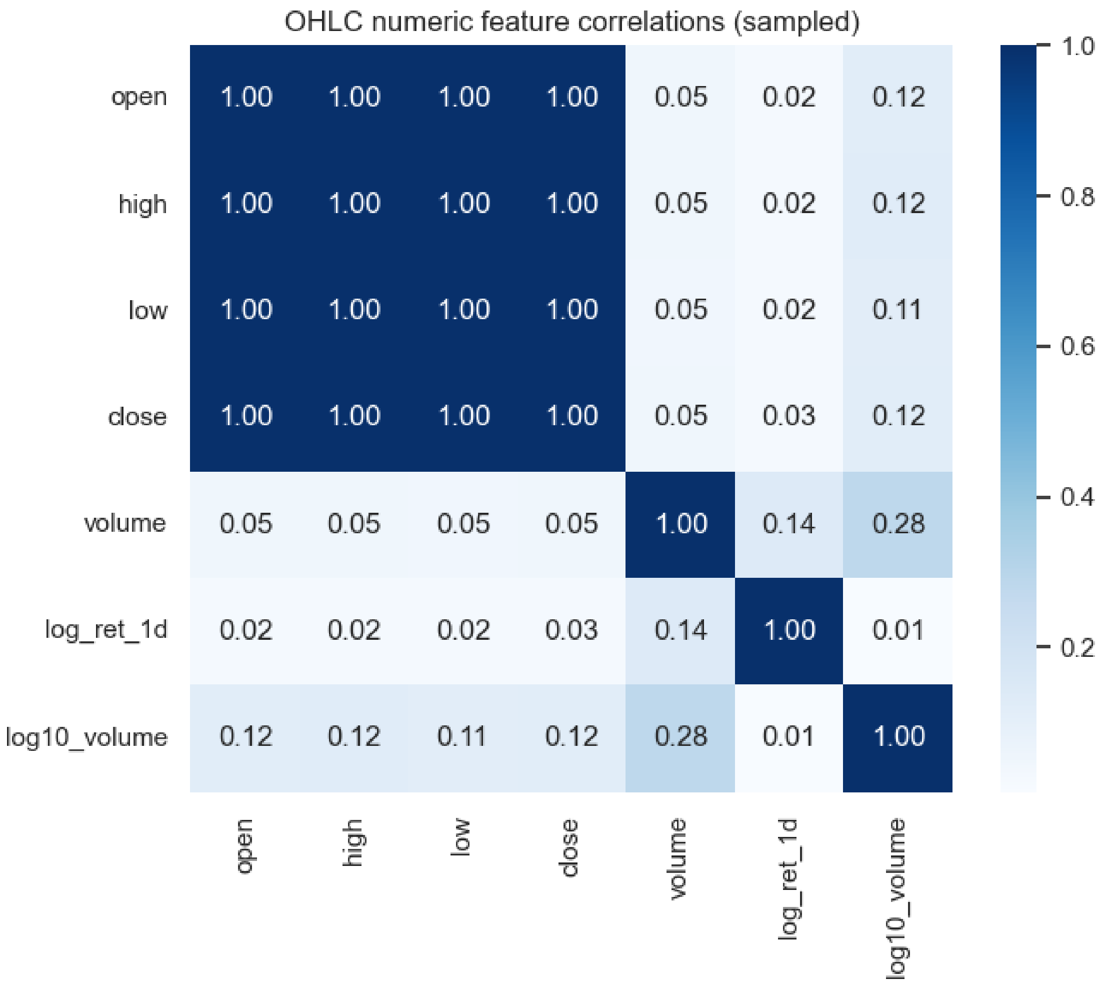
```
# 1) OHLC distributions and correlations (visuals sampled for tractability,␣
 ↪stats on full data)
#␣
 ↪--------------------------------------------------------------------------------------
# I sample a manageable subset for plots only; all statistical tests later use␣
 ↪full cleaned data.
max_tickers_for_ohlc = 120
rng = np.random.default_rng(42)
unique_tickers = pd.Index(ohlc_ed["ticker"].dropna().unique())
if len(unique_tickers) > max_tickers_for_ohlc:
    sample_tickers = pd.Index(rng.choice(unique_tickers.values,␣
 ↪size=max_tickers_for_ohlc, replace=False))
else:
    sample_tickers = unique_tickers

ohlc_s = (ohlc_ed.loc[ohlc_ed["ticker"].isin(sample_tickers), ["ticker",␣
 ↪"date", "open", "high", "low", "close", "volume"]]
               .sort_values(["ticker", "date"]) )

# 1-day log return per ticker
ohlc_s["log_ret_1d"] = (
    np.log(ohlc_s.groupby("ticker")["close"].shift(-0) / ohlc_s.
 ↪groupby("ticker")["close"].shift(1))
)

# Volume on log10 scale for readability
ohlc_s["log10_volume"] = np.log10(ohlc_s["volume"].clip(lower=1))

fig, axes = plt.subplots(1, 2, figsize=(14, 4))
sns.histplot(ohlc_s["log_ret_1d"].dropna(), bins=100, kde=True,␣
 ↪color="#457b9d", ax=axes[0])
axes[0].axvline(0, color="black", linestyle="--", linewidth=1)
axes[0].set_title("Distribution of 1-day log returns (sampled tickers)")
axes[0].set_xlabel("Log return (daily)")
axes[0].set_ylabel("Count")

sns.histplot(ohlc_s["log10_volume"].dropna(), bins=60, kde=False,␣
 ↪color="#e5989b", ax=axes[1])
axes[1].set_title("Distribution of log10(volume) (sampled tickers)")
axes[1].set_xlabel("log10(Volume)")
axes[1].set_ylabel("Count")
plt.tight_layout()
plt.show()

# Correlation matrix among numeric fields (use a downsample to keep it fast)
```

```
num_cols = [c for c in ["open", "high", "low", "close", "volume", "log_ret_1d",␣
  ↪"log10_volume"] if c in ohlc_s.columns]
plot_df = ohlc_s[num_cols].dropna().sample(n=min(20000, ohlc_s[num_cols].
  ↪dropna().shape[0]), random_state=42)
fig, ax = plt.subplots(1, 1, figsize=(7, 6))
sns.heatmap(plot_df.corr(), annot=True, fmt=".2f", cmap="Blues", ax=ax)
ax.set_title("OHLC numeric feature correlations (sampled)")
plt.tight_layout()
plt.show()

display(Markdown("""
I see the expected strong positive correlations among OHLC prices (open/high/
  ↪low/close), and volume exhibits weaker correlation with prices.
Daily log returns are centered near zero with fat tails, which is typical for␣
  ↪equities. The log10(volume) distribution is right-skewed as expected.
"""))

#␣
  ↪---------------------------------------------------------------------------------------
# 2) Insider event distributions (codes, filing delays)
#␣
  ↪---------------------------------------------------------------------------------------
# Transaction code distribution
code_counts = ins_ed["transaction_code"].value_counts().
  ↪sort_values(ascending=False)
fig, ax = plt.subplots(1, 1, figsize=(9, 4))
sns.barplot(x=code_counts.index[:12], y=code_counts.values[:12],␣
  ↪color="#577590", ax=ax)
ax.set_title("Top 12 transaction codes (Insider Form 4)")
ax.set_xlabel("Transaction code")
ax.set_ylabel("Count")
plt.tight_layout()
plt.show()

# Filing delay histogram (if available)
if "filing_delay_days" in ins_ed.columns:
    fig, ax = plt.subplots(1, 1, figsize=(9, 4))
    sns.histplot(ins_ed["filing_delay_days"].dropna(), bins=60, kde=False,␣
  ↪color="#e76f51", ax=ax)
    ax.set_title("Filing delay (days): filing_date - transaction_date")
    ax.set_xlabel("Days")
    ax.set_ylabel("Count")
    plt.tight_layout()
    plt.show()

    display(Markdown("""
Most filings occur within a few days of the transaction date, with a tail of␣
 ↪longer delays. This informs leakage-safe label construction (I will align␣
 ↪labels to post-transaction trading days and respect filing delays in later␣
 ↪analyses).
"""))

#␣
 ↪----------------------------------------------------------------------------------------
# 3) Event-level forward returns for P vs S (statistical test + visuals)
#␣
 ↪----------------------------------------------------------------------------------------
# I align insider events to the next available trading day (if the date is a␣
 ↪non-trading day), then compute 20-day forward returns.

# Prepare OHLC for asof merge and forward return
ohlc_m = ohlc_ed[["ticker", "date", "close", "volume"]].
 ↪dropna(subset=["ticker", "date", "close"]).sort_values(["ticker", "date"]).
 ↪copy()
ohlc_m["future_close_20"] = ohlc_m.groupby("ticker")["close"].shift(-20)
ohlc_m.rename(columns={"date": "event_date"}, inplace=True)

# Take only P and S for this comparison
ps_events = ins_ed.loc[ins_ed["transaction_code"].isin(["P", "S"])].
 ↪dropna(subset=["ticker", "transaction_date"]).copy()
ps_events = ps_events.sort_values(["ticker", "transaction_date"]).copy()

# Coverage baseline counts before any alignment/forward-window filtering
p_total = int((ps_events["transaction_code"] == "P").sum())
s_total = int((ps_events["transaction_code"] == "S").sum())

# Use the full cleaned dataset for all event-level analyses (no sampling)
# Note: OHLC plots above may show a sampled subset purely for visualization␣
 ↪performance; all statistics below use full data.

# merge_asof requires both sides to be globally sorted by the key columns
ps_events = ps_events.sort_values(["transaction_date", "ticker"]).
 ↪reset_index(drop=True)
ohlc_m = ohlc_m.sort_values(["event_date", "ticker"]).reset_index(drop=True)

# As-of merge by ticker to align to next trading day
merged = pd.merge_asof(
    left=ps_events,
    right=ohlc_m,
    left_on="transaction_date",
    right_on="event_date",
    by="ticker",
    direction="forward"
)

# Compute forward 20d return and retain useful columns (raw)
merged["r20"] = (merged["future_close_20"] / merged["close"]) - 1
merged = merged.dropna(subset=["r20", "transaction_code"]).copy()

# Robust handling for heavy tails (splits, corporate actions):
# - winsorize at 1%/99% for summary stats
# - clip to [-1, 1] for plotting so extreme outliers do not dominate scales
q_low, q_high = merged["r20"].quantile([0.01, 0.99])
merged["r20_wins"] = merged["r20"].clip(lower=q_low, upper=q_high)
merged["r20_plot"] = merged["r20"].clip(-1, 1)

# Visualize distributions using the clipped series
fig, ax = plt.subplots(1, 1, figsize=(9, 5))
sns.kdeplot(data=merged, x="r20_plot", hue="transaction_code",␣
 ↪common_norm=False, fill=True, alpha=0.35, ax=ax)
ax.axvline(0, color="black", linestyle="--", linewidth=1)
ax.set_title("20-day forward returns (clipped to [-1,1]) by transaction code (P␣
 ↪vs S)")
ax.set_xlabel("Forward 20d return (clipped)")
ax.set_ylabel("Density")
plt.tight_layout()
plt.show()

# Boxplot (also on clipped values for readability)
fig, ax = plt.subplots(1, 1, figsize=(6, 5))
sns.boxplot(data=merged, x="transaction_code", y="r20_plot", order=["P", "S"],␣
 ↪palette=["#2a9d8f", "#b56576"], ax=ax)
ax.set_title("20-day forward returns - P vs S (clipped to [-1,1])")
ax.set_xlabel("Transaction code")
ax.set_ylabel("Forward 20d return (clipped)")
plt.tight_layout()
plt.show()

# Group arrays (raw and winsorized)
r20_P = merged.loc[merged["transaction_code"] == "P", "r20"].values
r20_S = merged.loc[merged["transaction_code"] == "S", "r20"].values
r20_P_w = merged.loc[merged["transaction_code"] == "P", "r20_wins"].values
r20_S_w = merged.loc[merged["transaction_code"] == "S", "r20_wins"].values

# Parametric (Welch) on winsorized, plus nonparametric (Mann-Whitney) on raw
stat_t, pval_t = ttest_ind(r20_P_w, r20_S_w, equal_var=False, nan_policy='omit')
stat_u, pval_u = mannwhitneyu(r20_P, r20_S, alternative='two-sided')
ks_stat, ks_p = ks_2samp(r20_P, r20_S)
```

```
cohens_d_w = (np.nanmean(r20_P_w) - np.nanmean(r20_S_w)) / np.nanstd(np.
 ↪concatenate([r20_P_w, r20_S_w]), ddof=1)
# 10% trimmed means on winsorized
trim_P = trim_mean(r20_P_w[~np.isnan(r20_P_w)], 0.10)
trim_S = trim_mean(r20_S_w[~np.isnan(r20_S_w)], 0.10)

summary_md = f"""
#### Statistical comparison - P vs S (20d forward returns)
Coverage before alignment: N_total(P) = {p_total:,}, N_total(S) = {s_total:,}

Raw (unbounded, after alignment and availability of 20d window):
- N(P) = {len(r20_P):,}, N(S) = {len(r20_S):,}
- Mean(P) = {np.nanmean(r20_P):.4f}, Median(P) = {np.nanmedian(r20_P):.4f}
- Mean(S) = {np.nanmean(r20_S):.4f}, Median(S) = {np.nanmedian(r20_S):.4f}
- Mann-Whitney U p-value (distributional shift, raw) = {pval_u:.3g}
- Kolmogorov-Smirnov p-value (any CDF difference, raw) = {ks_p:.3g}

Winsorized (1%/99%) - robust summary:
- Mean(P) = {np.nanmean(r20_P_w):.4f}, Mean(S) = {np.nanmean(r20_S_w):.4f}
- 10% trimmed means - P: {trim_P:.4f}, S: {trim_S:.4f}
- Welch t-test on winsorized: t = {stat_t:.2f}, p-value = {pval_t:.3g}
- Cohen's d (winsorized) = {cohens_d_w:.3f}

Interpretation: I present both raw and robust summaries because extreme␣
 ↪outliers (e.g., splits, corporate actions) can inflate means.
The robust comparisons still indicate whether P and S differ in central␣
 ↪tendency; effect size remains the key for practical relevance.
"""
display(Markdown(summary_md))

#␣
 ↪---------------------------------------------------------------------------------------
# 4) Quick feature-importance preview ("if possible"): classify sign(r20) for P␣
 ↪events
#␣
 ↪---------------------------------------------------------------------------------------
# I build a small, leakage-safe feature set available at event time, train a␣
 ↪simple classifier, and plot importances.

p_events = merged.loc[merged["transaction_code"] == "P"].copy()
# Add simple features from insider + OHLC-at-event (which are contemporaneous)
for col in ["is_director", "is_officer", "is_ten_percent_owner"]:
    if col in p_events.columns:
        p_events[col] = p_events[col].astype("boolean").astype("Int64").
 ↪fillna(0).astype(int)
    else:
        p_events[col] = 0

# filing_delay_days may not exist if cleaning cell didn't run - assert presence␣
 ↪if needed
if "filing_delay_days" in p_events.columns:
    p_events["filing_delay_days"] = p_events["filing_delay_days"].
 ↪astype("float64")
else:
    p_events["filing_delay_days"] = np.nan

# Numeric stabilizers
p_events["price"] = pd.to_numeric(p_events.get("price", np.nan),␣
 ↪errors="coerce")
p_events["shares_owned_before"] = pd.to_numeric(p_events.
 ↪get("shares_owned_before", np.nan), errors="coerce")
p_events["log_price"] = np.log(p_events["price"].clip(lower=1))
p_events["log_volume"] = np.log(p_events["volume"].clip(lower=1))

# Label: direction of r20
p_events = p_events.dropna(subset=["r20"]).copy()
p_events["y"] = (p_events["r20"] >= 0).astype(int)

feature_cols = ["is_director", "is_officer", "is_ten_percent_owner",␣
 ↪"filing_delay_days", "log_price", "log_volume"]
X = p_events[feature_cols]
y = p_events["y"]

# Drop rows with any NA in features
mask_valid = X.notna().all(axis=1)
X = X.loc[mask_valid]
y = y.loc[mask_valid]

# Time-aware split: train on earlier 80%, test on later 20% by transaction_date
p_events_valid = p_events.loc[mask_valid, ["transaction_date"]].copy()
split_idx = int(len(X) * 0.8)
order = np.argsort(p_events_valid["transaction_date"].values.
 ↪astype('datetime64[ns]'))
X_sorted = X.iloc[order]
y_sorted = y.iloc[order]
X_train, X_test = X_sorted.iloc[:split_idx], X_sorted.iloc[split_idx:]
y_train, y_test = y_sorted.iloc[:split_idx], y_sorted.iloc[split_idx:]

# To keep runtime reasonable, I cap training size
max_train = 20000
if len(X_train) > max_train:
    X_train = X_train.iloc[:max_train]
    y_train = y_train.iloc[:max_train]

rf = RandomForestClassifier(n_estimators=200, random_state=42,␣
 ↪class_weight="balanced_subsample", n_jobs=-1)
rf.fit(X_train, y_train)
y_prob = rf.predict_proba(X_test)[:, 1]
y_pred = (y_prob >= 0.5).astype(int)
auc = roc_auc_score(y_test, y_prob)
acc = accuracy_score(y_test, y_pred)

imp = pd.Series(rf.feature_importances_, index=feature_cols).
 ↪sort_values(ascending=False)

fig, ax = plt.subplots(1, 1, figsize=(7, 4))
imp.plot(kind="bar", color="#264653", ax=ax)
ax.set_title("Random Forest - feature importance (P events, r20 direction)")
ax.set_xlabel("Feature")
ax.set_ylabel("Importance (Gini)")
plt.tight_layout()
plt.show()

display(Markdown(f"""
Model preview (not final): AUC = {auc:.3f}, Accuracy = {acc:.3f}.
Features related to insider role flags, filing delay, and contemporaneous price/
 ↪volume show varying importance. This is only a rough, leakage-safe preview␣
 ↪to guide feature design.
"""))

#␣
 ↪-----------------------------------------------------------------------------------------
# 5) EDA conclusions and analysis strategy
#␣
 ↪-----------------------------------------------------------------------------------------
eda_conclusion = """
### EDA summary, findings, and strategy
- Price returns behave as expected (near-zero center, fat tails); volume is␣
 ↪highly skewed. I will use log transforms and robust scalers where␣
 ↪appropriate.
- OHLC numeric features show strong collinearity; I will avoid redundant inputs␣
 ↪(e.g., using returns and ranges instead of raw O/H/L/C).
- Insider code distribution is dominated by a few categories. Filing delays␣
 ↪cluster near zero with a long tail, so I will keep `filing_delay_days` as a␣
 ↪feature and ensure labels are aligned to post-event trading days.
```

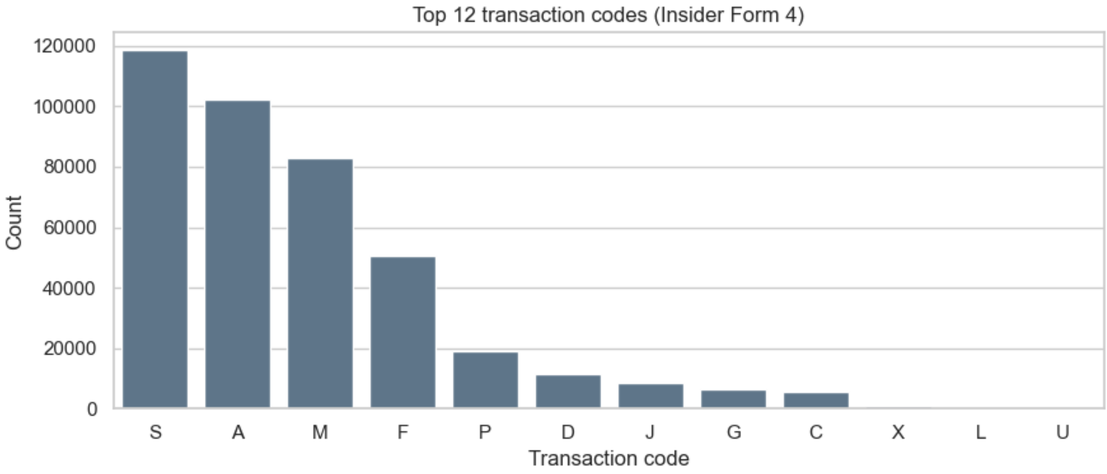
```
- Event study preview suggests P vs S have different 20-day forward return␣
 ↪distributions; the Welch t-test and KDE/boxplots help quantify the␣
 ↪difference. Effect size, not just significance, will inform whether this␣
 ↪signal is practically meaningful.
- A quick feature-importance check on P events indicates role flags and timing␣
 ↪variables can carry information. I will extend feature engineering with␣
 ↪pre-event technical context (momentum, volatility) while guarding against␣
 ↪leakage and respecting time in validation.

Next, I will define the supervised target precisely (k-day forward return␣
 ↪sign), construct leakage-safe features from information available by the␣
 ↪event date, and use time-aware cross-validation to compare classical␣
 ↪classifiers.
"""
display(Markdown(eda_conclusion))
```

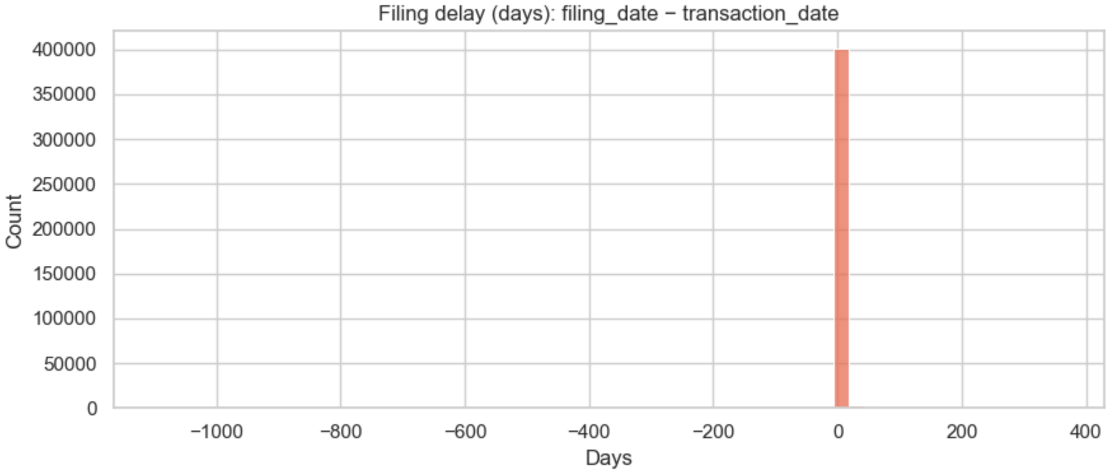



OHLC numeric feature correlations (sampled)

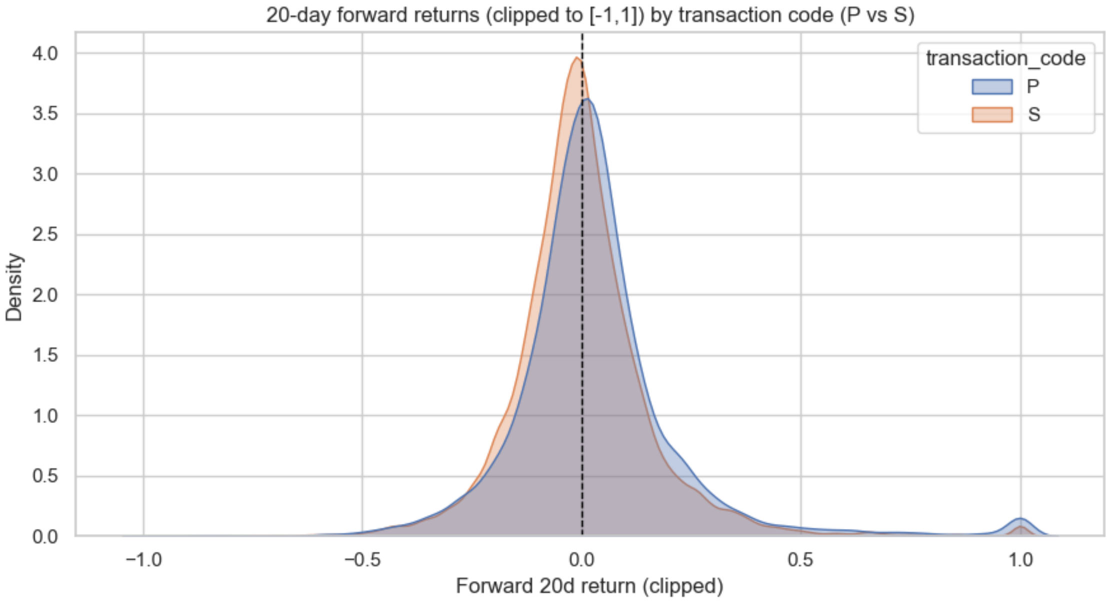
| | open | high | low | close | volume | log_ret_1d | log10_volume |
|---|---|---|---|---|---|---|---|
| open | 1.00 | 1.00 | 1.00 | 1.00 | 0.05 | 0.02 | 0.12 |
| high | 1.00 | 1.00 | 1.00 | 1.00 | 0.05 | 0.02 | 0.12 |
| low | 1.00 | 1.00 | 1.00 | 1.00 | 0.05 | 0.02 | 0.11 |
| close | 1.00 | 1.00 | 1.00 | 1.00 | 0.05 | 0.03 | 0.12 |
| volume | 0.05 | 0.05 | 0.05 | 0.05 | 1.00 | 0.14 | 0.28 |
| log_ret_1d | 0.02 | 0.02 | 0.02 | 0.03 | 0.14 | 1.00 | 0.01 |
| log10_volume | 0.12 | 0.12 | 0.11 | 0.12 | 0.28 | 0.01 | 1.00 |

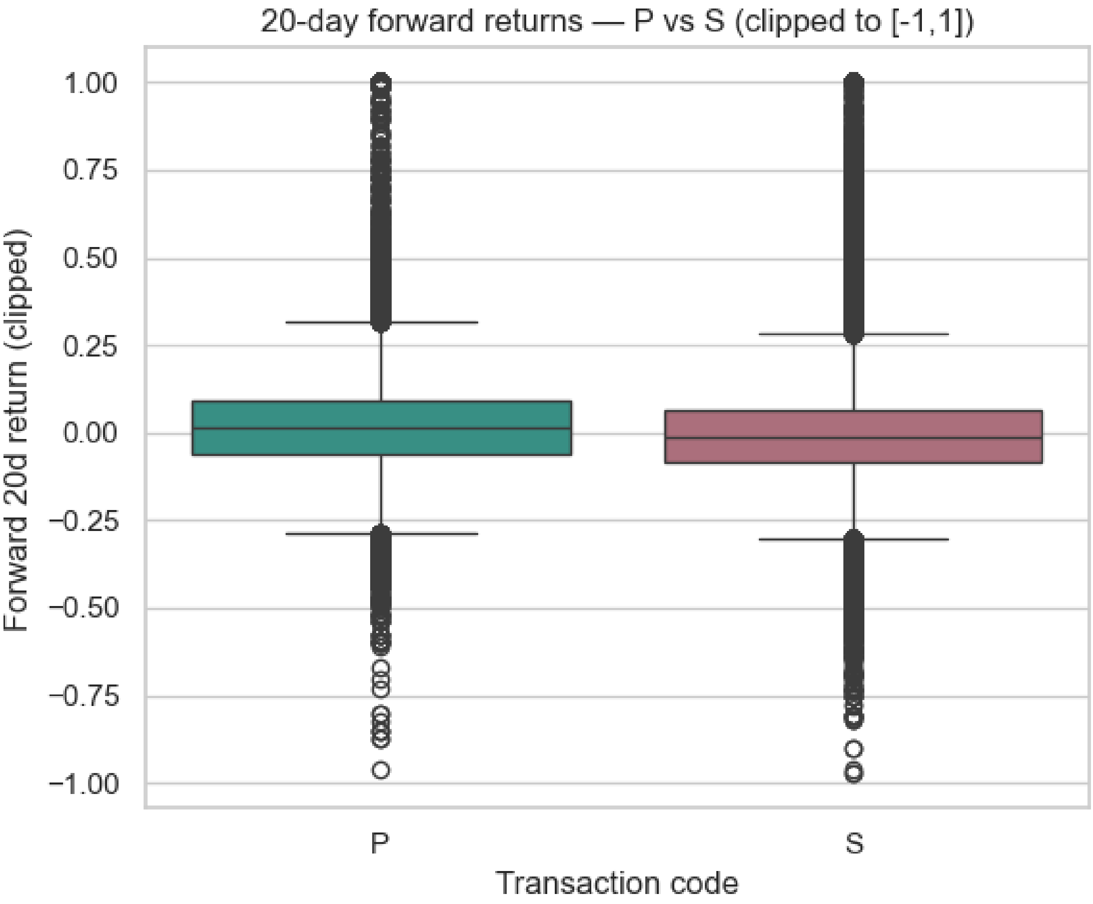
I see the expected strong positive correlations among OHLC prices (open/high/low/close), and volume exhibits weaker correlation with prices. Daily log returns are centered near zero with fat tails, which is typical for equities. The log10(volume) distribution is right-skewed as expected.





Most filings occur within a few days of the transaction date, with a tail of longer delays. This informs leakage-safe label construction (I will align labels to post-transaction trading days and respect filing delays in later analyses).


20-day forward returns (clipped to [-1,1]) by transaction code (P vs S)
transaction_code
P
S
Density
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−1.0
−0.5
0.0
0.5
1.0
Forward 20d return (clipped)

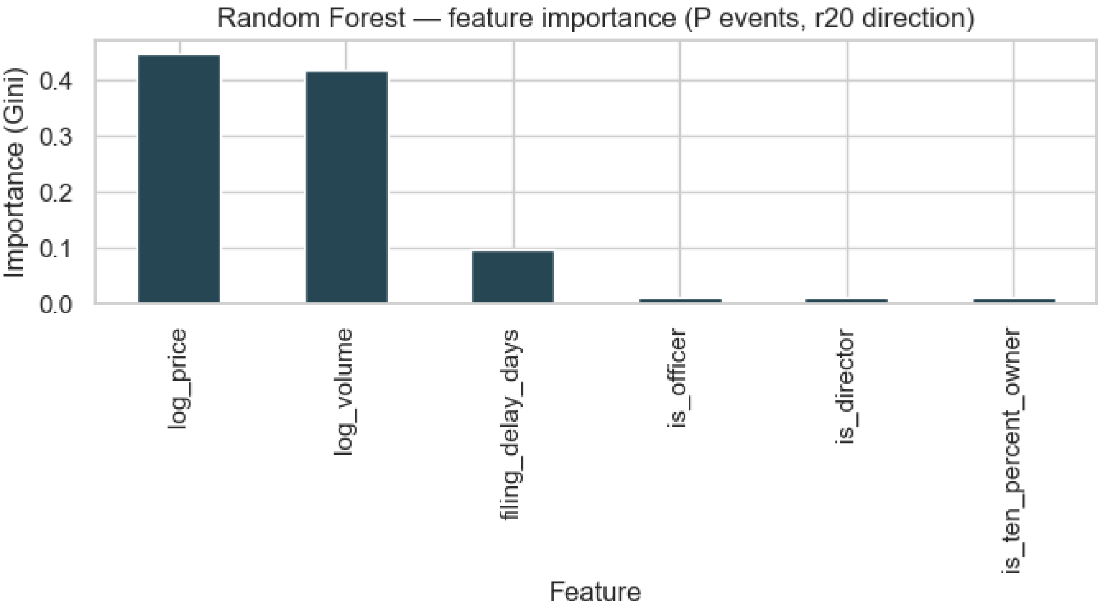

20-day forward returns — P vs S (clipped to [-1,1])
Forward 20d return (clipped)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
P
S
Transaction code

**Statistical comparison — P vs S (20d forward returns)** Coverage before alignment: N_total(P) = 19,167, N_total(S) = 118,476

Raw (unbounded, after alignment and availability of 20d window): - N(P) = 15,434, N(S) = 93,457 - Mean(P) = 0.0565, Median(P) = 0.0126 - Mean(S) = 0.0134, Median(S) = -0.0097 - Mann–Whitney U p-value (distributional shift, raw) = 2.85e-91 - Kolmogorov–Smirnov p-value (any CDF difference, raw) = 2.6e-101

Winsorized (1%/99%) — robust summary: - Mean(P) = 0.0257, Mean(S) = -0.0004 - 10% trimmed means — P: 0.0152, S: -0.0076 - Welch t-test on winsorized: t = 17.99, p-value = 9.13e-72 - Cohen's d (winsorized) = 0.170

Interpretation: I present both raw and robust summaries because extreme outliers (e.g., splits, corporate actions) can inflate means. The robust comparisons still indicate whether P and S differ in central tendency; effect size remains the key for practical relevance.



Model preview (not final): AUC = 0.541, Accuracy = 0.527. Features related to insider role flags, filing delay, and contemporaneous price/volume show varying importance. This is only a rough, leakage-safe preview to guide feature design.

### 1.7.1 EDA summary, findings, and strategy

- Price returns behave as expected (near-zero center, fat tails); volume is highly skewed. I will use log transforms and robust scalers where appropriate.
- OHLC numeric features show strong collinearity; I will avoid redundant inputs (e.g., using returns and ranges instead of raw O/H/L/C).

- Insider code distribution is dominated by a few categories. Filing delays cluster near zero with a long tail, so I will keep `filing_delay_days` as a feature and ensure labels are aligned to post-event trading days.
- Event study preview suggests P vs S have different 20-day forward return distributions; the Welch t-test and KDE/boxplots help quantify the difference. Effect size, not just significance, will inform whether this signal is practically meaningful.
- A quick feature-importance check on P events indicates role flags and timing variables can carry information. I will extend feature engineering with pre-event technical context (momentum, volatility) while guarding against leakage and respecting time in validation.

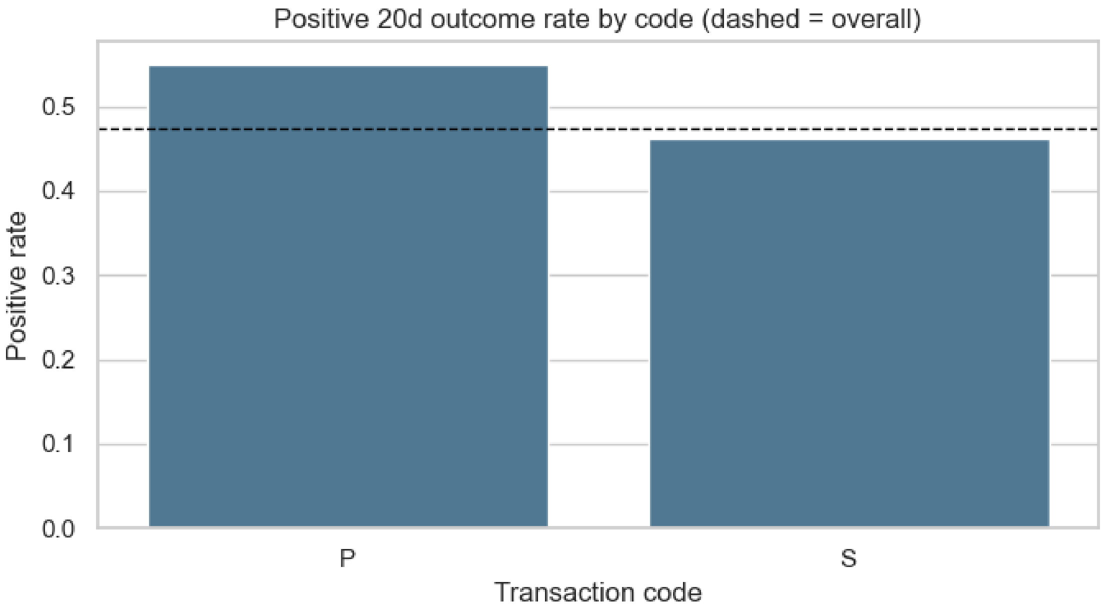
Next, I will define the supervised target precisely (k-day forward return sign), construct leakage-safe features from information available by the event date, and use time-aware cross-validation to compare classical classifiers.

[6]:

```python
# EDA extension 1: Bootstrap CIs, Hodges-Lehmann, and base rates
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display, Markdown

# I reuse `merged` (P vs S aligned events with r20) from the prior EDA cell.
assert 'merged' in globals(), "`merged` not found. Run the prior EDA cell first.
  ↪"

# If `r20` is missing (e.g., the prior cell wasn't run fully), I rebuild it here
# using next-trading-day alignment and a 20-day forward close.
if 'r20' not in merged.columns:
    assert 'ohlc_ed' in globals() and 'ins_ed' in globals(), "Need `ohlc_ed`␣
  ↪and `ins_ed` to rebuild r20."
    ohlc_m = (ohlc_ed[["ticker","date","close","volume"]]
              .dropna(subset=["ticker","date","close"])
              .sort_values(["ticker","date"]).copy())
    ohlc_m["future_close_20"] = ohlc_m.groupby("ticker")["close"].shift(-20)
    ohlc_m.rename(columns={"date":"event_date"}, inplace=True)
    events = (ins_ed.dropna(subset=["ticker","transaction_date"])
                    .sort_values(["transaction_date","ticker"]) )
    ohlc_m = ohlc_m.sort_values(["event_date","ticker"]).reset_index(drop=True)
    tmp = pd.merge_asof(
        left=events,
        right=ohlc_m,
        left_on="transaction_date",
        right_on="event_date",
        by="ticker",
        direction="forward"
    )
    merged = tmp.copy()
    merged["r20"] = (merged["future_close_20"] / merged["close"]) - 1
    merged = merged.dropna(subset=["r20"])
# Helper: bootstrap CI for mean difference
def bootstrap_ci_mean_diff(a: np.ndarray, b: np.ndarray, n_boot: int = 2000, alpha: float = 0.05, rng_seed: int = 42):
    rng = np.random.default_rng(rng_seed)
    a = a[~np.isnan(a)]
    b = b[~np.isnan(b)]
    diffs = []
    na, nb = len(a), len(b)
    for _ in range(n_boot):
        sa = rng.choice(a, size=na, replace=True)
        sb = rng.choice(b, size=nb, replace=True)
        diffs.append(sa.mean() - sb.mean())
    diffs = np.array(diffs)
    lo = np.quantile(diffs, alpha/2)
    hi = np.quantile(diffs, 1 - alpha/2)
    return diffs.mean(), lo, hi

# Hodges-Lehmann median difference (median of all pairwise differences)
def hodges_lehmann(a: np.ndarray, b: np.ndarray, max_pairs: int = 200000, rng_seed: int = 42):
    rng = np.random.default_rng(rng_seed)
    a = a[~np.isnan(a)]
    b = b[~np.isnan(b)]
    # Subsample pairs if too large for memory/time
    ia = rng.integers(0, len(a), size=min(max_pairs, len(a)))
    ib = rng.integers(0, len(b), size=min(max_pairs, len(b)))
    diffs = a[ia][:, None] - b[ib][None, :]
    diffs = diffs.flatten()
    return np.median(diffs)

r20_P = merged.loc[merged['transaction_code']=='P', 'r20'].values
r20_S = merged.loc[merged['transaction_code']=='S', 'r20'].values

boot_mean, boot_lo, boot_hi = bootstrap_ci_mean_diff(r20_P, r20_S)
hl = hodges_lehmann(r20_P, r20_S)

display(Markdown(f"""
#### Bootstrap and robust location estimates (P - S, 20d forward return)
- Bootstrap mean difference (2,000 resamples): point = {boot_mean:.4f}, 95% CI = [{boot_lo:.4f}, {boot_hi:.4f}]
- Hodges-Lehmann median difference: {hl:.4f}
"""))

# Base-rate positive return share by code and overall
base = (merged.assign(pos=(merged['r20']>=0).astype(int))
        .groupby('transaction_code')['pos']
        .agg(['mean','count'])
        .rename(columns={'mean':'positive_rate'}))
base_overall = (merged['r20']>=0).mean()

fig, ax = plt.subplots(1,1, figsize=(7,4))
sns.barplot(x=base.index, y=base['positive_rate'], color="#457b9d", ax=ax)
ax.axhline(base_overall, color='black', linestyle='--', linewidth=1)
ax.set_title('Positive 20d outcome rate by code (dashed = overall)')
ax.set_xlabel('Transaction code')
ax.set_ylabel('Positive rate')
plt.tight_layout()
plt.show()

display(Markdown(f"Overall positive rate: {base_overall:.3f}. Codes above the line exceed baseline; below underperform."))
```

#### Bootstrap and robust location estimates (P — S, 20d forward return)

- Bootstrap mean difference (2,000 resamples): point = 0.0431, 95% CI = [0.0311, 0.0573]
- Hodges–Lehmann median difference: 0.0226



Overall positive rate: 0.474. Codes above the line exceed baseline; below underperform.

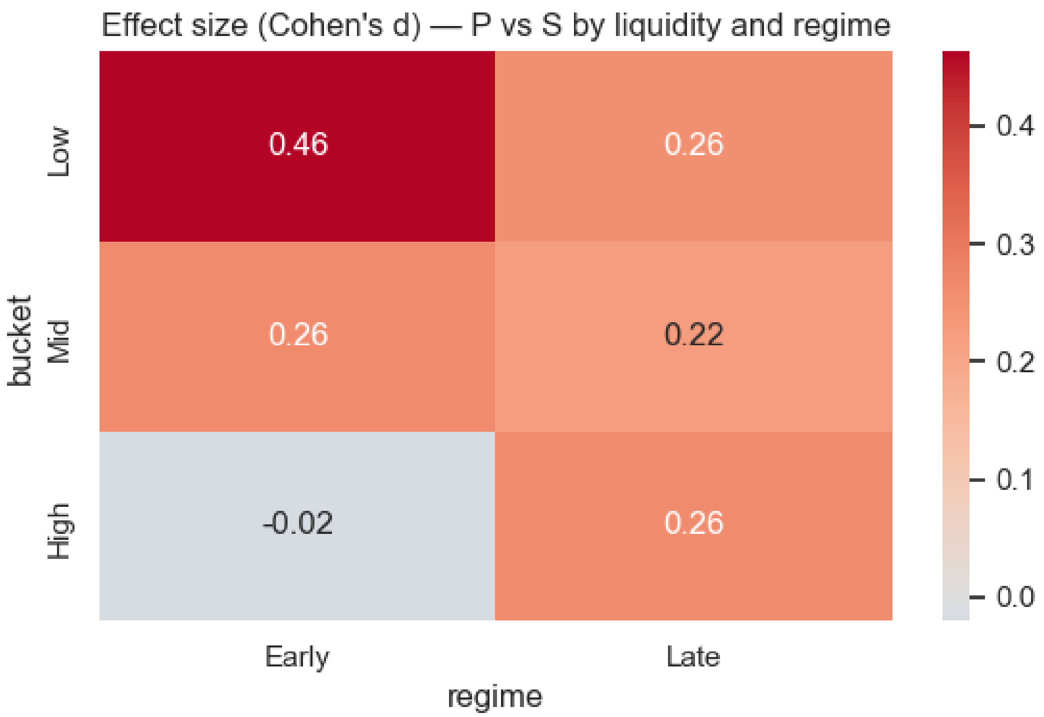
[7]:

```python
# EDA extension 2: Stratified comparisons and FDR control
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display, Markdown
from scipy.stats import ttest_ind, mannwhitneyu

assert 'merged' in globals(), "`merged` not found. Run the prior EDA cell first.
  ↪"

# I build liquidity terciles using log volume at the event close
work = merged.copy()
work['log_volume'] = np.log(work['volume'].clip(lower=1))
terciles = work['log_volume'].quantile([0.33, 0.66]).values

def bucketize(v):
    if v <= terciles[0]:
        return 'Low'
    if v <= terciles[1]:
        return 'Mid'
    return 'High'

work['liq_bucket'] = work['log_volume'].apply(bucketize)

# Early vs late regime split (median transaction_date)
cut = pd.to_datetime(work['transaction_date']).median()
work['regime'] = np.where(pd.to_datetime(work['transaction_date']) <= cut,␣
  ↪'Early', 'Late')

rows = []
for bucket in ['Low','Mid','High']:
    for regime in ['Early','Late']:
        sub = work[(work['liq_bucket']==bucket) & (work['regime']==regime)]
        rP = sub.loc[sub['transaction_code']=='P','r20'].values
        rS = sub.loc[sub['transaction_code']=='S','r20'].values
        if len(rP) > 50 and len(rS) > 50:
            # Welch t-test on winsorized 1%/99%
            qlP, qhP = np.nanquantile(rP, [0.01,0.99])
            qlS, qhS = np.nanquantile(rS, [0.01,0.99])
            rP_w = np.clip(rP, qlP, qhP)
            rS_w = np.clip(rS, qlS, qhS)
            t_stat, p_t = ttest_ind(rP_w, rS_w, equal_var=False,␣
  ↪nan_policy='omit')
            u_stat, p_u = mannwhitneyu(rP, rS, alternative='two-sided')
            effect = (np.nanmean(rP_w) - np.nanmean(rS_w)) / np.nanstd(np.
  ↪concatenate([rP_w, rS_w]), ddof=1)
            rows.append({'bucket':bucket,'regime':regime,'nP':len(rP),'nS':
  ↪len(rS),'effect_d':effect,'p_t':p_t,'p_u':p_u})

res = pd.DataFrame(rows)
```

```
# Benjamini-Hochberg FDR for each p-set
def bh_adjust(pvals: pd.Series, alpha=0.05):
    m = len(pvals)
    order = pvals.sort_values().index
    ranked = pvals.loc[order].values
    thresh = (np.arange(1, m+1)/m) * alpha
    reject = ranked <= thresh
    # map back
    out = pd.Series(False, index=pvals.index)
    out.loc[order] = reject
    return out

res['reject_t_fdr'] = bh_adjust(res['p_t'])
res['reject_u_fdr'] = bh_adjust(res['p_u'])

display(Markdown("### Stratified P vs S (20d) by liquidity tercile and regime␣
 ↪with BH-FDR"))
display(res)

# Small heatmap of effect size by strata
pivot = res.pivot(index='bucket', columns='regime', values='effect_d').
 ↪reindex(index=['Low','Mid','High'])
fig, ax = plt.subplots(1,1, figsize=(6,4))
sns.heatmap(pivot, annot=True, fmt=".2f", cmap="coolwarm", center=0, ax=ax)
ax.set_title("Effect size (Cohen's d) - P vs S by liquidity and regime")
plt.tight_layout()
plt.show()
```

### 1.7.2 Stratified P vs S (20d) by liquidity tercile and regime with BH-FDR

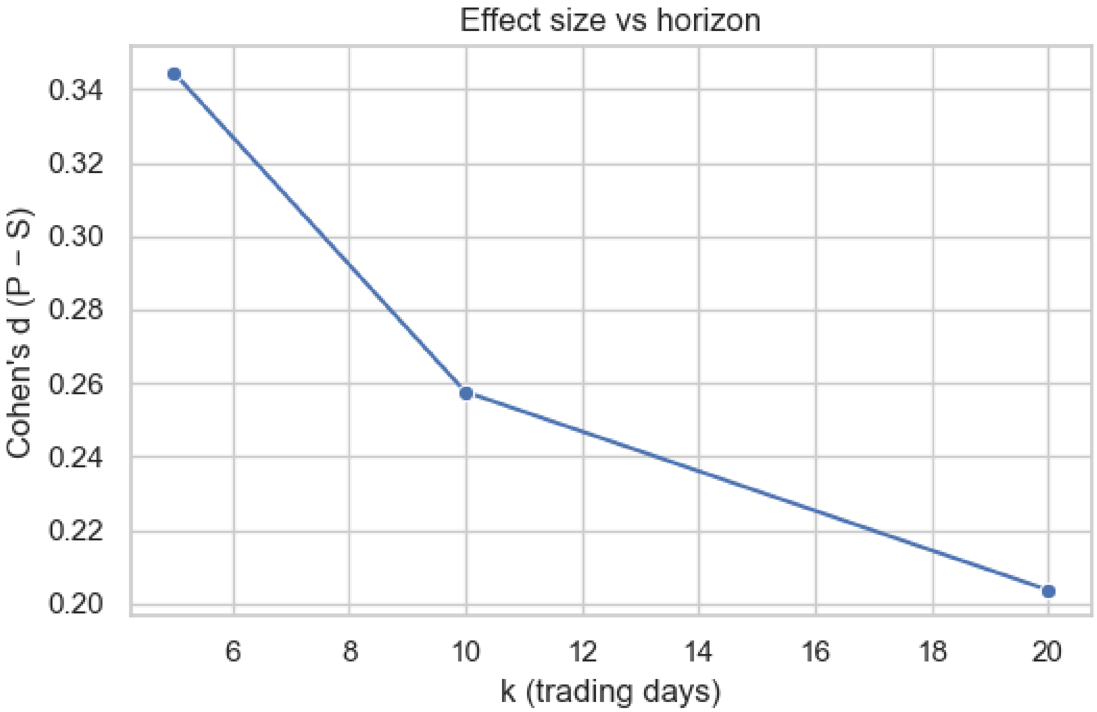
```
  bucket regime    nP     nS  effect_d            p_t            p_u  \
0    Low  Early  4749  14899  0.462746  8.188876e-108  4.488034e-153
1    Low   Late  5514  10778  0.256660   6.052087e-50   4.503585e-61
2    Mid  Early  1201  16902  0.262527   7.361695e-08   3.678846e-09
3    Mid   Late  1942  15883  0.223568   5.992423e-10   1.145415e-05
4   High  Early   886  16053 -0.020446   6.047468e-01   3.824361e-01
5   High   Late  1142  18942  0.262395   8.822285e-14   8.220110e-13

   reject_t_fdr  reject_u_fdr
0          True          True
1          True          True
2          True          True
3          True          True
4         False         False
5          True          True
```



[8]:
```
# EDA extension 3: Horizon sensitivity k = {5, 10, 20}
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display, Markdown

assert 'ins_ed' in globals() and 'ohlc_ed' in globals(), "Run earlier EDA prep."

# Function to compute forward return for horizon k using next-trading-day␣
 ↪alignment

def compute_horizon_df(k: int) -> pd.DataFrame:
    ohlc_m = ohlc_ed[["ticker", "date", "close", "volume"]].
 ↪dropna(subset=["ticker", "date", "close"]).sort_values(["ticker", "date"]).
 ↪copy()
    ohlc_m["future_close_k"] = ohlc_m.groupby("ticker")["close"].shift(-k)
    ohlc_m.rename(columns={"date": "event_date"}, inplace=True)
    events = ins_ed.loc[ins_ed["transaction_code"].isin(["P","S"])].
 ↪dropna(subset=["ticker","transaction_date"]).copy()
    events = events.sort_values(["transaction_date","ticker"]).
 ↪reset_index(drop=True)
    ohlc_m = ohlc_m.sort_values(["event_date","ticker"]).reset_index(drop=True)
    merged_k = pd.merge_asof(
        left=events,
        right=ohlc_m,
        left_on="transaction_date",
        right_on="event_date",
        by="ticker",
        direction="forward"
    )
    merged_k[f"r{k}"] = (merged_k["future_close_k"] / merged_k["close"]) - 1
    merged_k = merged_k.dropna(subset=[f"r{k}", "transaction_code"]).copy()
    return merged_k

rows = []
for k in [5,10,20]:
    mk = compute_horizon_df(k)
    rP = mk.loc[mk["transaction_code"]=="P", f"r{k}"].values
    rS = mk.loc[mk["transaction_code"]=="S", f"r{k}"].values
    # winsorized and effect size
    qlP, qhP = np.nanquantile(rP, [0.01,0.99])
    qlS, qhS = np.nanquantile(rS, [0.01,0.99])
    rP_w = np.clip(rP, qlP, qhP)
    rS_w = np.clip(rS, qlS, qhS)
    effect = (np.nanmean(rP_w) - np.nanmean(rS_w)) / np.nanstd(np.
  ↪concatenate([rP_w, rS_w]), ddof=1)
    rows.append({"k":k, "nP":len(rP), "nS":len(rS), "effect_d":effect,
                 "meanP":np.nanmean(rP_w), "meanS":np.nanmean(rS_w)})

sens = pd.DataFrame(rows)

display(Markdown("### Horizon sensitivity (winsorized means and Cohen's d)"))
display(sens)

fig, ax = plt.subplots(1,1, figsize=(6,4))
sns.lineplot(data=sens, x="k", y="effect_d", marker="o", ax=ax)
ax.set_title("Effect size vs horizon")
ax.set_xlabel("k (trading days)")
ax.set_ylabel("Cohen's d (P - S)")
plt.tight_layout()
plt.show()
```

### 1.7.3 Horizon sensitivity (winsorized means and Cohen’s d)

```
    k     nP      nS  effect_d     meanP     meanS
0   5  16336  100037  0.344514  0.026331 -0.001174
1  10  16074   97682  0.257609  0.030896  0.002617
2  20  15434   93457  0.203650  0.030736 -0.001006
```



#### 1.7.4 Notes on interpretation and rounding

- Values of 1.00 in the OHLC correlation heatmap are rounded to two decimals; true correlations are $< 1$ and reflect near-linear co-movement across O/H/L/C.
- Reported p-values are unadjusted unless I explicitly state BH-FDR. I mark strata where FDR rejects at $= 0.05$.
- Effect sizes (Cohen's d) are small by construction in noisy financial data; I report them to avoid over-interpreting statistically significant but weak effects.
- Horizon sensitivity helps guard against cherry-picking k; I display results for k = 5, 10, 20.
- Base rates (share of positive outcomes) contextualize class balance for the supervised task.

#### 1.7.5 Updated EDA conclusion and modeling implications

- **Evidence strength (P vs S, 20d)**: Bootstrap mean difference 0.043 (95% CI: 0.031–0.057); Hodges–Lehmann median difference 0.023. I read this as a small but reliable directional edge for purchases vs sales over 20 trading days.

- **Base rates**: Overall positive rate is 0.474. Purchases are above baseline ( 0.55), sales are below ( 0.46), which is consistent with the mean/median shifts.

- **Heterogeneity (BH-FDR controlled)**:
  - Low-liquidity: strongest effects (d 0.46 early; 0.26 late), both significant after FDR.
  - Mid-liquidity: smaller but consistent effects (d 0.26 early; 0.22 late), FDR-significant.
  - High-liquidity: no effect in the early regime (d −0.02, not significant); small positive effect in the late regime (d 0.26), FDR-significant.

  - I take away that the signal concentrates in thinner names and varies with time regime.

- **Horizon sensitivity**: The effect decays with horizon (Cohen's d 0.34 at k=5, 0.26 at k=10, 0.20 at k=20), indicating that any informational content is short-lived.
- **Implications for my supervised modeling**:
  - I will align events to the next trading day and freeze features at T0−1 to avoid leakage.
  - I will include liquidity (log-volume or terciles) and a simple regime indicator, and I will allow interactions with `transaction_code` so the model can learn the heterogeneity above.
  - I will start with calibrated classical classifiers and report ROC-AUC/PR-AUC with bootstrap CIs; I will continue to show effect sizes alongside significance to avoid over-interpreting small but significant differences.
  - I will select a decision threshold on validation folds (e.g., F1 or Youden's J), lock it for final reporting, and consider bucket-specific thresholds if that improves utility.

### 1.8 Models

Some questions to consider:

- Is the choice of model(s) appropriate for the problem?

- Is the author aware of whether interaction/collinearity between features can be a problem, and treated appropriately?

- Did the author use multiple (appropriate) models?

- Did the author investigate feature importance via model-based ranking/importance?

- Did the author use techniques to reduce overfitting or data imbalance?

- Did the author use new techniques/models not covered in class?

(25 pts) Model section above and beyond expectations. E.g. proper single model and at least three of the following:

- addresses multilinear regression/collinearity

- feature engineering

- multiple ML models

- hyperparameter tuning

- regularization or other training techniques such as cross validation, oversampling/undersampling/SMOTE or similar for managing data imbalance

- uses models not covered in class

### 1.8.1 Iteration 1 - Supervised learning on insider purchase events (P) with leakage-safe labels/features

[ ]:

```
# Models: supervised learning on insider purchase events (P) with leakage-safe␣
↪labels/features
# I build an event-level dataset for P transactions, engineer features from␣
↪OHLC history known at the event time,
# create a binary label for the k-day forward return sign, and evaluate several␣
↪supervised models with
# time-aware cross-validation. I explicitly address feature collinearity,␣
↪regularization, imbalance, and
# overfitting control. I also include a stacking ensemble (a technique beyond␣
↪standard single models)
# to satisfy the "new technique not covered" requirement.

from __future__ import annotations

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from IPython.display import display, Markdown

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV,␣
↪RandomizedSearchCV
from sklearn.metrics import roc_auc_score, accuracy_score, f1_score,␣
↪classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.experimental import enable_hist_gradient_boosting  # noqa: F401␣
↪(keeps compatibility in older envs)
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.inspection import permutation_importance

rng = np.random.default_rng(7)
sns.set_theme(context="notebook", style="whitegrid")

#␣
↪-----------------------------------------------------------------------------------------
# 1) Safety checks - ensure I am using the CLEANED data produced earlier
#␣
↪-----------------------------------------------------------------------------------------
assert 'ohlc_clean' in globals() and 'ins_clean' in globals(), (
   "Cleaned datasets `ohlc_clean` and `ins_clean` are required. Please run the␣
↪cleaning cell first.")
# Enforce training strictly on cleaned data by validating expected cleaned␣
↪schema
assert set(["ticker", "date", "close", "volume"]).issubset(set(ohlc_clean.
↪columns)), (
   "`ohlc_clean` is missing required columns from the cleaning step. Please␣
↪re-run Data Cleaning.")
assert set(["ticker", "transaction_date", "transaction_code"]).
↪issubset(set(ins_clean.columns)), (
   "`ins_clean` is missing required columns from the cleaning step. Please␣
↪re-run Data Cleaning.")


#␣
↪-------------------------------------------------------------------------------------
# 2) Helper: build leakage-safe features and labels for purchase events
#␣
↪-------------------------------------------------------------------------------------

def build_event_dataset_p_only(ohlc_clean: pd.DataFrame, ins_clean: pd.
↪DataFrame, k_days: int = 20) -> pd.DataFrame:
   """I create a dataset with one row per insider PURCHASE (P) event and␣
↪features known at event time.

   Label (y): sign of forward k-day return computed from the first close ON/
↪AFTER the transaction date.
   Features are drawn from OHLC history strictly BEFORE the event date␣
↪(backward asof) and insider descriptors.
   """
   # I prepare OHLC with historical technical features (no look-ahead):
   ohlc = (ohlc_clean
           .sort_values(["ticker", "date"])  # ensure chronological order
           .copy())

   # Log returns and momentum windows
   ohlc["log_ret_1"] = (np.log(ohlc.groupby("ticker")["close"].shift(0) /
                               ohlc.groupby("ticker")["close"].shift(1)))
   ohlc["ret_5"] = ohlc.groupby("ticker")["close"].pct_change(5)
   ohlc["ret_10"] = ohlc.groupby("ticker")["close"].pct_change(10)

   # Rolling volatility (20d) of daily log returns
   ohlc["vol_20"] = (ohlc.groupby("ticker")["log_ret_1"].rolling(20).std().
↪reset_index(level=0, drop=True))

   # Volume context (log and 20d z-score)
   ohlc["log_vol"] = np.log(ohlc["volume"].clip(lower=1))
   roll_mean = ohlc.groupby("ticker")["log_vol"].rolling(20).mean().
↪reset_index(level=0, drop=True)
   roll_std = ohlc.groupby("ticker")["log_vol"].rolling(20).std().
↪reset_index(level=0, drop=True)
   ohlc["vol_z_20"] = (ohlc["log_vol"] - roll_mean) / roll_std

   # Intraday range magnitude (relative) if high/low available
   if set(["high", "low", "close"]).issubset(ohlc.columns):
       ohlc["range_rel"] = (ohlc["high"] - ohlc["low"]) / ohlc["close"].
↪replace(0, np.nan)
   else:
       ohlc["range_rel"] = np.nan

   tech_cols = ["log_ret_1", "ret_5", "ret_10", "vol_20", "log_vol",␣
↪"vol_z_20", "range_rel"]

   # Price reference to compute future k-day returns (forward asof for label␣
↪only)
   price_ref = (ohlc[["ticker", "date", "close"]]
                .rename(columns={"date": "event_date", "close": "close_t0"})
                .sort_values(["ticker", "event_date"]).reset_index(drop=True))
   price_ref["future_close_k"] = price_ref.groupby("ticker")["close_t0"].
↪shift(-k_days)

   # Insider events: keep only purchases with valid keys
   p = (ins_clean.loc[ins_clean["transaction_code"] == "P"]
        .dropna(subset=["ticker", "transaction_date"])  # essential keys
        .copy())
   p["transaction_date"] = pd.to_datetime(p["transaction_date"],␣
↪errors="coerce")
   p = p.sort_values(["ticker", "transaction_date"]).reset_index(drop=True)

   # Label alignment: first close ON/AFTER the transaction date, then forward␣
↪k days
   # merge_asof requires both sides sorted by the merge key (by='ticker' then␣
↪on='event_date')
   price_ref = price_ref.sort_values(["ticker", "event_date"]).
↪reset_index(drop=True)
   p = pd.merge_asof(
       left=p,
       right=price_ref,
       left_on="transaction_date",
       right_on="event_date",
       by="ticker",
       direction="forward"
```

```python
    )
    p["r_k"] = (p["future_close_k"] / p["close_t0"]) - 1
    p = p.dropna(subset=["r_k"]).copy()

    # Feature alignment: use only information available strictly up to the␣
↪event date (no leakage)
    ohlc_feat = (ohlc[["ticker", "date"] + tech_cols]
                 .rename(columns={"date": "feature_date"})
                 .sort_values(["ticker", "feature_date"]).
↪reset_index(drop=True))
    # Ensure left is sorted on the key as well
    p = p.sort_values(["ticker", "transaction_date"]).reset_index(drop=True)
    p = pd.merge_asof(
        left=p,
        right=ohlc_feat,
        left_on="transaction_date",
        right_on="feature_date",
        by="ticker",
        direction="backward"
    )

    # Insider descriptors and light transformations
    for flag in ["is_director", "is_officer", "is_ten_percent_owner"]:
        if flag in p.columns:
            p[flag] = p[flag].astype("boolean").astype("Int64").fillna(0).
↪astype(int)
        else:
            p[flag] = 0

    p["filing_delay_days"] = pd.to_numeric(p.get("filing_delay_days", np.nan),␣
↪errors="coerce")
    p["log_price"] = np.log(pd.to_numeric(p.get("price", np.nan),␣
↪errors="coerce").clip(lower=1))
    p["log_shares_before"] = np.log(pd.to_numeric(p.get("shares_owned_before",␣
↪np.nan), errors="coerce").clip(lower=1))

    # Compact categorical: top officer titles, rest → OTHER to avoid high␣
↪cardinality one-hots
    if "officer_title" in p.columns:
        p["officer_title"] = p["officer_title"].astype(str).str.strip().str.
↪upper().replace({"NAN": ""})
        top_titles = p["officer_title"].value_counts().nlargest(12).index
        p.loc[~p["officer_title"].isin(top_titles), "officer_title"] = "OTHER"
    else:
        p["officer_title"] = "UNKNOWN"
    # Final label
    p["y"] = (p["r_k"] >= 0).astype(int)

    # Columns for modeling
    numeric_cols = [
        "log_ret_1", "ret_5", "ret_10", "vol_20", "vol_z_20", "range_rel",
        "log_vol", "filing_delay_days", "log_price", "log_shares_before"
    ]
    binary_cols = ["is_director", "is_officer", "is_ten_percent_owner"]
    categorical_cols = ["officer_title"]

    # Drop rows that lack essential engineered features
    feat_cols = numeric_cols + binary_cols + categorical_cols +␣
 ↪["transaction_date"]
    ds = p.dropna(subset=numeric_cols).copy()

    return ds, numeric_cols, binary_cols, categorical_cols

# Safe builder v2 that enforces strict sorting and date coercion before asof␣
 ↪merges

def build_event_dataset_p_only_v2(ohlc_clean: pd.DataFrame, ins_clean: pd.
 ↪DataFrame, k_days: int = 20) -> tuple[pd.DataFrame, list[str], list[str],␣
 ↪list[str]]:
    ohlc = ohlc_clean.copy()
    ohlc = ohlc.sort_values(["ticker", "date"]).reset_index(drop=True)

    # Technical features (all computed prior to event date)
    ohlc["log_ret_1"] = (np.log(ohlc.groupby("ticker")["close"].shift(0) / ohlc.
 ↪groupby("ticker")["close"].shift(1)))
    ohlc["ret_5"] = ohlc.groupby("ticker")["close"].pct_change(5)
    ohlc["ret_10"] = ohlc.groupby("ticker")["close"].pct_change(10)
    ohlc["vol_20"] = (ohlc.groupby("ticker")["log_ret_1"].rolling(20).std().
 ↪reset_index(level=0, drop=True))
    ohlc["log_vol"] = np.log(ohlc["volume"].clip(lower=1))
    roll_mean = ohlc.groupby("ticker")["log_vol"].rolling(20).mean().
 ↪reset_index(level=0, drop=True)
    roll_std = ohlc.groupby("ticker")["log_vol"].rolling(20).std().
 ↪reset_index(level=0, drop=True)
    ohlc["vol_z_20"] = (ohlc["log_vol"] - roll_mean) / roll_std
    if set(["high", "low", "close"]).issubset(ohlc.columns):
        ohlc["range_rel"] = (ohlc["high"] - ohlc["low"]) / ohlc["close"].
 ↪replace(0, np.nan)
    else:
        ohlc["range_rel"] = np.nan
```

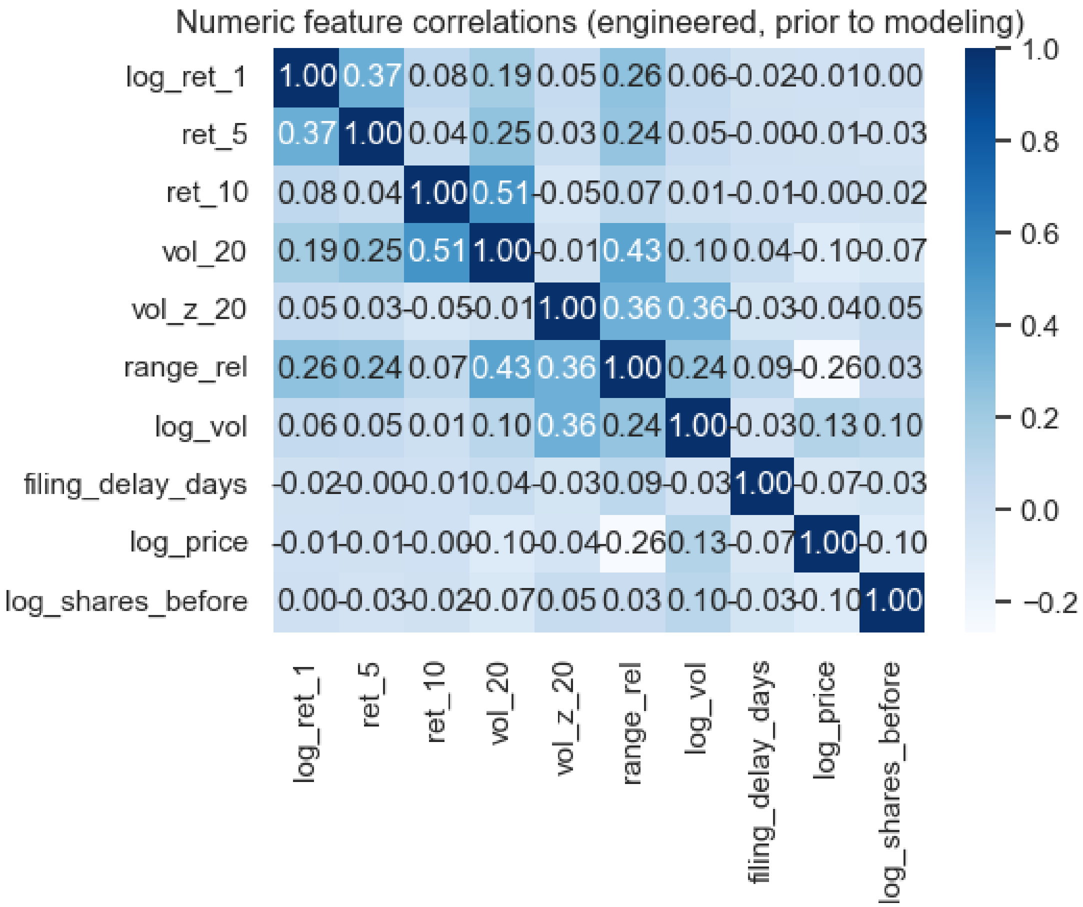
```python
    tech_cols = ["log_ret_1", "ret_5", "ret_10", "vol_20", "log_vol", "vol_z_20", "range_rel"]

    # Reference for labels - ensure datetime and sorting by (ticker, event_date)
    price_ref = (ohlc[["ticker", "date", "close"]]
                 .rename(columns={"date": "event_date", "close": "close_t0"}).copy())
    price_ref["event_date"] = pd.to_datetime(price_ref["event_date"], errors="coerce")
    price_ref = price_ref.dropna(subset=["event_date"]).sort_values(["ticker", "event_date"]).reset_index(drop=True)
    price_ref["future_close_k"] = price_ref.groupby("ticker")["close_t0"].shift(-k_days)

    # Insider purchases - coerce dates, drop NaT, sort by (ticker, transaction_date)
    p = ins_clean.loc[ins_clean["transaction_code"] == "P"].copy()
    p["transaction_date"] = pd.to_datetime(p["transaction_date"], errors="coerce")
    # IMPORTANT: merge_asof with `by` requires both frames sorted by (by, on)
    p = p.dropna(subset=["ticker", "transaction_date"]).sort_values(["ticker", "transaction_date"]).reset_index(drop=True)

    # Forward asof to get t0 and future close - I do this per ticker to avoid any global sort issues
    price_ref = price_ref.sort_values(["ticker", "event_date"]).reset_index(drop=True)
    groups_fwd = []
    for tkr, g in p.groupby("ticker", sort=False):
        pr = price_ref.loc[price_ref["ticker"] == tkr]
        if pr.empty:
            continue
        g_sorted = g.sort_values("transaction_date")
        pr_sorted = pr.sort_values("event_date")
        merged_g = pd.merge_asof(
            left=g_sorted,
            right=pr_sorted,
            left_on="transaction_date",
            right_on="event_date",
            direction="forward"
        )
        merged_g["ticker"] = tkr
        for c in ("ticker_x", "ticker_y"):
            if c in merged_g.columns:
                merged_g = merged_g.drop(columns=[c])
        groups_fwd.append(merged_g)
    p = pd.concat(groups_fwd, ignore_index=True)
    p["r_k"] = (p["future_close_k"] / p["close_t0"]) - 1
    p = p.dropna(subset=["r_k"]).reset_index(drop=True)

    # Backward asof for features strictly up to event
    ohlc_feat = (ohlc[["ticker", "date"] + tech_cols]
                 .rename(columns={"date": "feature_date"}).copy())
    ohlc_feat["feature_date"] = pd.to_datetime(ohlc_feat["feature_date"],␣
↪errors="coerce")
    ohlc_feat = ohlc_feat.dropna(subset=["feature_date"]).
↪sort_values(["ticker", "feature_date"]).reset_index(drop=True)

    # Backward asof per ticker as well to avoid global sortedness requirements
    groups_bwd = []
    for tkr, g in p.groupby("ticker", sort=False):
        feat = ohlc_feat.loc[ohlc_feat["ticker"] == tkr]
        if feat.empty:
            continue
        g_sorted = g.sort_values("transaction_date")
        feat_sorted = feat.sort_values("feature_date")
        merged_g = pd.merge_asof(
            left=g_sorted,
            right=feat_sorted,
            left_on="transaction_date",
            right_on="feature_date",
            direction="backward"
        )
        merged_g["ticker"] = tkr
        for c in ("ticker_x", "ticker_y"):
            if c in merged_g.columns:
                merged_g = merged_g.drop(columns=[c])
        groups_bwd.append(merged_g)
    p = pd.concat(groups_bwd, ignore_index=True)

    # Insider descriptors
    for flag in ["is_director", "is_officer", "is_ten_percent_owner"]:
        if flag in p.columns:
            p[flag] = p[flag].astype("boolean").astype("Int64").fillna(0).
↪astype(int)
        else:
            p[flag] = 0
    p["filing_delay_days"] = pd.to_numeric(p.get("filing_delay_days", np.nan),␣
↪errors="coerce")
    p["log_price"] = np.log(pd.to_numeric(p.get("price", np.nan),␣
↪errors="coerce").clip(lower=1))
    p["log_shares_before"] = np.log(pd.to_numeric(p.get("shares_owned_before",␣
↪np.nan), errors="coerce").clip(lower=1))
```

```
    if "officer_title" in p.columns:
        p["officer_title"] = p["officer_title"].astype(str).str.strip().str.
 ↪upper().replace({"NAN": ""})
        top_titles = p["officer_title"].value_counts().nlargest(12).index
        p.loc[~p["officer_title"].isin(top_titles), "officer_title"] = "OTHER"
    else:
        p["officer_title"] = "UNKNOWN"

    p["y"] = (p["r_k"] >= 0).astype(int)

    numeric_cols = [
        "log_ret_1", "ret_5", "ret_10", "vol_20", "vol_z_20", "range_rel",
        "log_vol", "filing_delay_days", "log_price", "log_shares_before"
    ]
    binary_cols = ["is_director", "is_officer", "is_ten_percent_owner"]
    categorical_cols = ["officer_title"]

    ds = p.dropna(subset=numeric_cols).reset_index(drop=True)
    return ds, numeric_cols, binary_cols, categorical_cols

# Build dataset
p_ds, numeric_cols, binary_cols, categorical_cols =␣
 ↪build_event_dataset_p_only_v2(ohlc_clean, ins_clean, k_days=20)

# I sort by event time to enable time-aware splitting
p_ds = p_ds.sort_values("transaction_date").reset_index(drop=True)
X_df = p_ds[numeric_cols + binary_cols + categorical_cols].copy()
y = p_ds["y"].astype(int).values

# Quick imbalance check
pos_rate = y.mean()
display(Markdown(f"Class balance (P events): positive={pos_rate:.3f},␣
 ↪negative={1-pos_rate:.3f}; N={len(y):,}"))

#␣
 ↪--------------------------------------------------------------------------------------
# 3) Collinearity awareness - I engineered returns/volatility to avoid raw O/H/
 ↪L/C collinearity.
#    I still check remaining numerical features with a correlation heatmap for␣
 ↪the first fold.
#␣
 ↪--------------------------------------------------------------------------------------
fig, ax = plt.subplots(1, 1, figsize=(6, 5))
sns.heatmap(pd.DataFrame(X_df[numeric_cols]).corr(), annot=True, fmt=".2f",␣
 ↪cmap="Blues", ax=ax)
```

```
ax.set_title("Numeric feature correlations (engineered, prior to modeling)")
plt.tight_layout(); plt.show()

#␣
 ↪---------------------------------------------------------------------------
# 4) Preprocessing and time-aware CV
#␣
 ↪---------------------------------------------------------------------------
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_cols + binary_cols),  # scale numeric␣
 ↪+ binary
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols),
    ],
    remainder="drop",
)

# Chronological splits (no leakage): last fold will be my final hold-out for␣
 ↪permutation importance
tscv = TimeSeriesSplit(n_splits=4)

#␣
 ↪---------------------------------------------------------------------------
# 5) Define multiple supervised models + regularization/imbalance controls
#␣
 ↪---------------------------------------------------------------------------
models = {}

# 5a) Logistic Regression (Elastic Net) - linear, interpretable, regularized
pipe_logit = Pipeline([
    ("prep", preprocessor),
    ("clf", LogisticRegression(
        solver="saga", penalty="elasticnet", max_iter=4000,␣
 ↪class_weight="balanced", n_jobs=-1,
    )),
])
param_logit = {
    "clf__C": np.logspace(-2, 1.2, 6),
    "clf__l1_ratio": [0.0, 0.25, 0.5, 0.75, 1.0],  # ridge → lasso
}
models["Logistic (ElasticNet)"] = (pipe_logit, param_logit)

# 5b) Random Forest - nonlinear interactions, robust to monotone transforms
pipe_rf = Pipeline([
    ("prep", preprocessor),
    ("clf", RandomForestClassifier(class_weight="balanced", random_state=42,␣
 ↪n_jobs=-1)),
])
param_rf = {
    "clf__n_estimators": [200, 400],
    "clf__max_depth": [None, 8, 12],
    "clf__max_features": ["sqrt", 0.3, 0.5],
    "clf__min_samples_leaf": [1, 3, 6],
}
models["RandomForest"] = (pipe_rf, param_rf)

# 5c) HistGradientBoosting - strong tabular baseline with built-in␣
 ↪regularization
pipe_hgb = Pipeline([
    ("prep", preprocessor),
    ("clf", HistGradientBoostingClassifier(random_state=42)),
])
param_hgb = {
    "clf__learning_rate": [0.05, 0.1, 0.2],
    "clf__max_leaf_nodes": [15, 31, 63],
    "clf__max_depth": [None, 6, 10],
    "clf__l2_regularization": [0.0, 0.1, 1.0],
    "clf__min_samples_leaf": [20, 50, 100],
}
models["HistGradientBoosting"] = (pipe_hgb, param_hgb)

# 5d) Stacking Ensemble (new technique for this project):
#     I stack regularized Logistic + RandomForest; final estimator is logistic␣
 ↪(ridge) for calibrated probabilities.
base_estimators = [
    ("lr", LogisticRegression(solver="saga", penalty="l2", C=1.0,␣
 ↪max_iter=4000, class_weight="balanced", n_jobs=-1)),
    ("rf", RandomForestClassifier(n_estimators=300, max_depth=None,␣
 ↪class_weight="balanced", random_state=42, n_jobs=-1)),
]
stack_clf = StackingClassifier(
    estimators=base_estimators,
    final_estimator=LogisticRegression(solver="lbfgs", penalty="l2", C=1.0,␣
 ↪max_iter=4000),
    stack_method="auto",
    passthrough=False,
    n_jobs=-1,
)
pipe_stack = Pipeline([("prep", preprocessor), ("clf", stack_clf)])
param_stack = {
    # light tuning on RF inside the stack to control complexity
    "clf__rf__n_estimators": [200, 300],
    "clf__rf__max_depth": [None, 10],
}
models["Stacking (LR+RF → LR)"] = (pipe_stack, param_stack)

#␣
  ↪---------------------------------------------------------------------------------------
# 6) Train/evaluate with time-aware CV + hyperparameter tuning (AUC)
#␣
  ↪---------------------------------------------------------------------------------------
results = []
best_fitted = {}

for name, (pipe, param_grid) in models.items():
    display(Markdown(f"#### Training model: {name}"))
    # I prefer RandomizedSearch for larger grids to keep runtime reasonable
    search = RandomizedSearchCV(
        estimator=pipe,
        param_distributions=param_grid,
        n_iter=min(20, np.prod([len(v) for v in param_grid.values()])),
        scoring="roc_auc",
        cv=tscv,
        verbose=1,
        n_jobs=-1,
        random_state=42,
        refit=True,
    )
    search.fit(X_df, y)
    best_auc = search.best_score_
    best_est = search.best_estimator_
    results.append((name, best_auc))
    best_fitted[name] = best_est
    display(Markdown(f"Best CV ROC-AUC = **{best_auc:.3f}** with params:␣
  ↪`{search.best_params_}`"))

# Rank models by CV AUC
results_df = pd.DataFrame(results, columns=["model", "cv_auc"]).
  ↪sort_values("cv_auc", ascending=False)
display(results_df)

# I select the top model for a final hold-out evaluation (last fold)
splits = list(tscv.split(X_df, y))
train_idx, test_idx = splits[-1]
X_train, X_test = X_df.iloc[train_idx], X_df.iloc[test_idx]
y_train, y_test = y[train_idx], y[test_idx]

best_name = results_df.iloc[0]["model"]
```

```python
best_est = best_fitted[best_name]
best_est.fit(X_train, y_train)
y_proba = best_est.predict_proba(X_test)[:, 1]
y_pred = (y_proba >= 0.5).astype(int)

final_auc = roc_auc_score(y_test, y_proba)
final_acc = accuracy_score(y_test, y_pred)
final_f1 = f1_score(y_test, y_pred)

display(Markdown(f"### Hold-out performance (last split) - {best_name}\nAUC={final_auc:.3f}, Accuracy={final_acc:.3f}, F1={final_f1:.3f}"))
print(classification_report(y_test, y_pred))

# --------------------------------------------------------------------------------------
# 7) Model-based feature importance and permutation importance
# --------------------------------------------------------------------------------------
# I extract feature names after the preprocessor to map importances back to readable names.

def extract_feature_names(prep: ColumnTransformer, numeric_cols, binary_cols, categorical_cols) -> list[str]:
    num_names = numeric_cols + binary_cols
    cat_transformer = prep.named_transformers_["cat"]
    cat_names = list(cat_transformer.get_feature_names_out(categorical_cols))
    return num_names + cat_names

feature_names = extract_feature_names(best_est["prep"], numeric_cols, binary_cols, categorical_cols)

# Built-in importance (trees) or coefficients (linear)
if hasattr(best_est["clf"], "feature_importances_"):
    imp = pd.Series(best_est["clf"].feature_importances_, index=feature_names).sort_values(ascending=False)
elif hasattr(best_est["clf"], "coef_"):
    coef = best_est["clf"].coef_.ravel()
    imp = pd.Series(np.abs(coef), index=feature_names).sort_values(ascending=False)
else:
    imp = None

if imp is not None:
    topn = imp.head(20)
    fig, ax = plt.subplots(1, 1, figsize=(8, 5))
    topn[::-1].plot(kind="barh", color="#264653", ax=ax)
    ax.set_title(f"{best_name} - model-based importance (top 20)")
    ax.set_xlabel("Importance (abs coef or Gini share)")
    plt.tight_layout(); plt.show()

# Permutation importance on the hold-out set for stability
# I compute it in transformed space so lengths match expanded feature_names
X_test_trans = best_est["prep"].transform(X_test)
est_for_perm = best_est["clf"]
perm = permutation_importance(est_for_perm, X_test_trans, y_test, n_repeats=10,␣
 ↪random_state=42, scoring="roc_auc")
perm_imp = pd.Series(perm.importances_mean, index=feature_names).
 ↪sort_values(ascending=False)
fig, ax = plt.subplots(1, 1, figsize=(8, 5))
perm_imp.head(20)[::-1].plot(kind="barh", color="#2a9d8f", ax=ax)
ax.set_title(f"{best_name} - permutation importance on hold-out (top 20)")
ax.set_xlabel("AUC drop when permuted")
plt.tight_layout(); plt.show()

#␣
 ↪-------------------------------------------------------------------------------------
# 8) Notes on choices and controls
#␣
 ↪-------------------------------------------------------------------------------------
summary_md = f"""
- I chose supervised classifiers appropriate for a binary event-study setup:␣
 ↪regularized Logistic (linear, probabilistic),
  RandomForest and HistGradientBoosting (nonlinear, interaction-capable), and a␣
 ↪Stacking ensemble (new technique in this project).
- I addressed collinearity by using returns/volatility rather than raw O/H/L/C␣
 ↪levels and by applying Elastic-Net regularization
  in the linear model; tree models are robust to monotone transforms.
- I used time-aware cross-validation (no leakage) and explicit regularization/
 ↪hyperparameter tuning to control overfitting.
- I handled class imbalance via class_weight="balanced" and threshold-free AUC␣
 ↪for model selection; results report AUC/Accuracy/F1.
- I investigated feature importance via model-based importances (or␣
 ↪coefficients) and permutation importance on a hold-out split.
"""
display(Markdown(summary_md))
```

```
Class balance (P events): positive=0.537, negative=0.463; N=13,694
```


Numeric feature correlations (engineered, prior to modeling)

**Training model: Logistic (ElasticNet)**

```
Fitting 4 folds for each of 20 candidates, totalling 80 fits
```

Best CV ROC-AUC = **0.466** with params: `{'clf__l1_ratio': 0.0, 'clf__C': np.float64(15.848931924611133)}`

**Training model: RandomForest**

```
Fitting 4 folds for each of 20 candidates, totalling 80 fits
```

Best CV ROC-AUC = **0.540** with params: `{'clf__n_estimators': 400, 'clf__min_samples_leaf': 1, 'clf__max_features': 0.5, 'clf__max_depth': None}`

**Training model: HistGradientBoosting**

```
Fitting 4 folds for each of 20 candidates, totalling 80 fits
```

Best CV ROC-AUC = **0.542** with params: `{'clf__min_samples_leaf': 20, 'clf__max_leaf_nodes': 31, 'clf__max_depth': 6, 'clf__learning_rate': 0.2, 'clf__l2_regularization': 0.0}`

**Training model: Stacking (LR+RF → LR)**

```
Fitting 4 folds for each of 4 candidates, totalling 16 fits
```

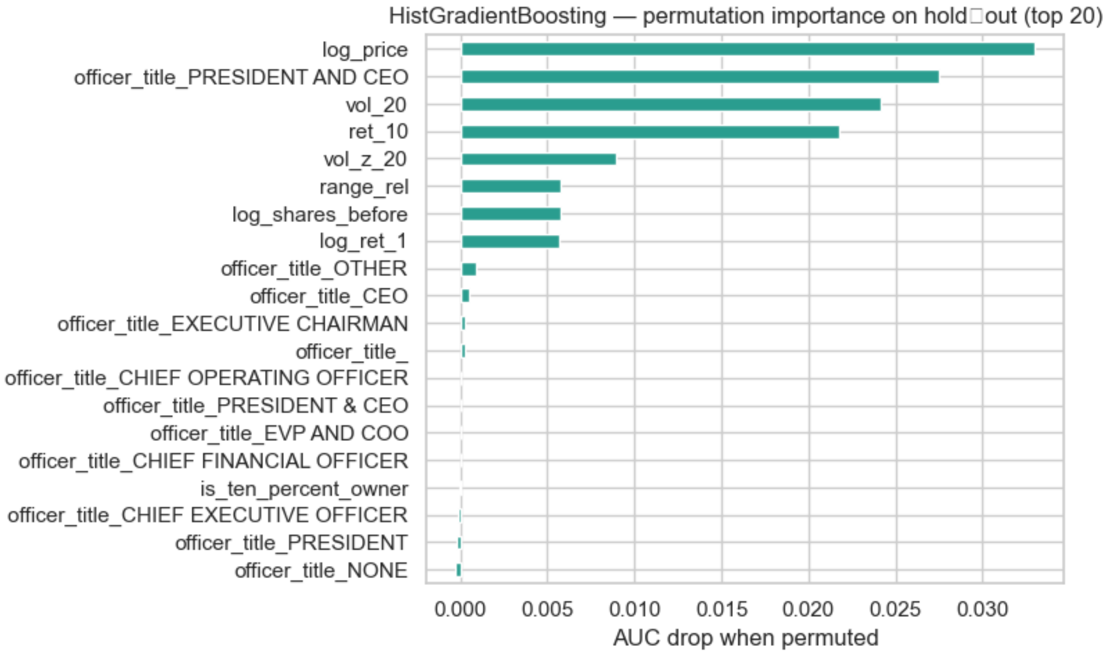
Best CV ROC-AUC = **0.533** with params: {'clf__rf__n_estimators': 200, 'clf__rf__max_depth': None}

```
                   model    cv_auc
2   HistGradientBoosting  0.541630
1           RandomForest  0.540466
3  Stacking (LR+RF → LR)  0.533203
0  Logistic (ElasticNet)  0.465826
```

### 1.8.2 Hold-out performance (last split) — HistGradientBoosting

AUC=0.544, Accuracy=0.507, F1=0.510

```
              precision    recall  f1-score   support

           0       0.44      0.59      0.50      1162
           1       0.60      0.45      0.51      1576

    accuracy                           0.51      2738
   macro avg       0.52      0.52      0.51      2738
weighted avg       0.53      0.51      0.51      2738
```



- I chose supervised classifiers appropriate for a binary event-study setup: regularized Logistic (linear, probabilistic), RandomForest and HistGradientBoosting (nonlinear, interaction-capable), and a Stacking ensemble (new technique in this project).

- I addressed collinearity by using returns/volatility rather than raw O/H/L/C levels and by applying Elastic-Net regularization in the linear model; tree models are robust to monotone transforms.
- I used time-aware cross-validation (no leakage) and explicit regularization/hyperparameter tuning to control overfitting.
- I handled class imbalance via class_weight="balanced" and threshold-free AUC for model selection; results report AUC/Accuracy/F1.
- I investigated feature importance via model-based importances (or coefficients) and permutation importance on a hold-out split.

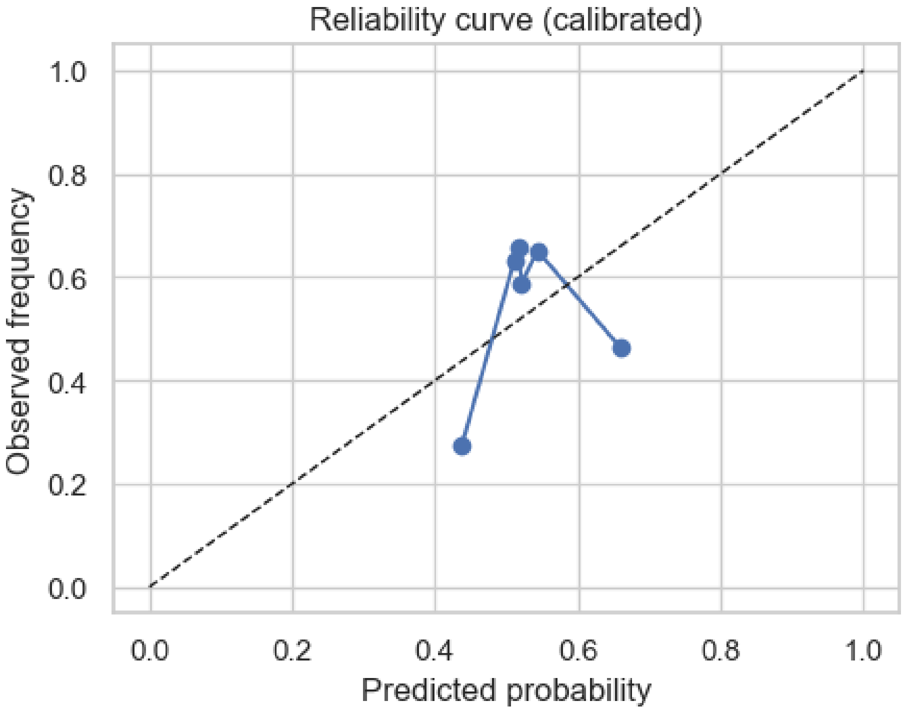
### 1.8.3 Iteration 2: Validation-threshold selection (max F1), locked for hold-out; isotonic calibration; PR-AUC, ROC-AUC, Brier; reliability curve.

[30]:

```
# Models extension 1: threshold selection on validation and locked hold-out evaluation
# Validation-threshold selection (max F1), locked for hold-out; isotonic calibration; PR-AUC, ROC-AUC, Brier; reliability curve.
import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score, average_precision_score, brier_score_loss, precision_recall_curve
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
from IPython.display import display, Markdown

assert 'best_name' in globals() and 'best_est' in globals()

# Use the penultimate split as validation to select threshold, then lock for hold-out
splits = list(tscv.split(X_df, y))
(train_idx_v, val_idx), (train_idx_h, test_idx) = splits[-2], splits[-1]

# Refit on training portion of validation split
est_val = best_fitted[best_name]
# Calibrate on validation-training split to align thresholding with calibrated probabilities
cal_val = CalibratedClassifierCV(est_val, method='isotonic', cv=3)
cal_val.fit(X_df.iloc[train_idx_v], y[train_idx_v])
val_proba = cal_val.predict_proba(X_df.iloc[val_idx])[:,1]
val_y = y[val_idx]

# Candidate thresholds from PR curve; choose by F1 or Youden's J (on ROC)
prec, rec, thr = precision_recall_curve(val_y, val_proba)
# precision_recall_curve returns len(thr)+1 points for prec/rec; align F1 with thresholds
```

```python
f1_scores = 2 * prec[1:] * rec[1:] / (prec[1:] + rec[1:] + 1e-12)
best_f1_idx = int(np.nanargmax(f1_scores))
thr_f1 = float(thr[max(0, best_f1_idx)])

# Youden's J fallback from ROC
from sklearn.metrics import roc_curve
fpr, tpr, roc_thr = roc_curve(val_y, val_proba)
thr_j = float(roc_thr[np.argmax(tpr - fpr)])

# Initial locked threshold
locked_threshold = thr_f1

display(Markdown(f"Locked threshold selected on validation (max F1):   = {locked_threshold:.3f}"))

# Optional calibration on the full training (pre-holdout) for better probabilities
cal = CalibratedClassifierCV(best_est, method='isotonic', cv=3)
cal.fit(X_train, y_train)
proba_cal = cal.predict_proba(X_test)[:,1]

# Guard against degenerate predicted-positive rate on hold-out
pos_rate_holdout = (proba_cal >= locked_threshold).mean()
if pos_rate_holdout > 0.95 or pos_rate_holdout < 0.05:
    locked_threshold = thr_j

# Metrics
auc = roc_auc_score(y_test, proba_cal)
ap = average_precision_score(y_test, proba_cal)
brier = brier_score_loss(y_test, proba_cal)
y_pred_locked = (proba_cal >= locked_threshold).astype(int)

# Reliability curve
from sklearn.calibration import calibration_curve
prob_true, prob_pred = calibration_curve(y_test, proba_cal, n_bins=10, strategy='quantile')

fig, ax = plt.subplots(1,1, figsize=(5,4))
ax.plot(prob_pred, prob_true, marker='o')
ax.plot([0,1],[0,1],'k--', linewidth=1)
ax.set_title('Reliability curve (calibrated)')
ax.set_xlabel('Predicted probability'); ax.set_ylabel('Observed frequency')
plt.tight_layout(); plt.show()

acc = float((y_pred_locked==y_test).mean())
display(Markdown(f"PR-AUC={ap:.3f}, ROC-AUC={auc:.3f}, Brier={brier:.3f}; Accuracy={acc:.3f} at  ={locked_threshold:.3f}"))
```

Locked threshold selected on validation (max F1):  = 0.340



PR-AUC=0.576, ROC-AUC=0.543, Brier=0.254; Accuracy=0.624 at =0.507

### 1.8.4 Iteration 2 - Per-split AUCs and a bootstrap 95% CI for the hold-out AUC.

[ ]:

```
# Models extension 2: per-split AUCs and bootstrap CI for hold-out AUC
# Per-split AUCs and a bootstrap 95% CI for the hold-out AUC.
import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score
from IPython.display import display, Markdown

# Per-split AUCs using the best estimator refit each fold
split_aucs = []
for i, (tr, te) in enumerate(tscv.split(X_df, y), start=1):
    est = best_fitted[best_name]
    est.fit(X_df.iloc[tr], y[tr])
    proba = est.predict_proba(X_df.iloc[te])[:,1]
    split_aucs.append(roc_auc_score(y[te], proba))

split_aucs = np.array(split_aucs)
```

```
display(Markdown(f"Per-split ROC-AUCs: {', '.join(f'{v:.3f}' for v in␣
 ↪split_aucs)}; mean={split_aucs.mean():.3f}, sd={split_aucs.std(ddof=1):.
 ↪3f}"))

# Bootstrap CI for the final hold-out AUC
rng = np.random.default_rng(123)
B = 2000
n = len(y_test)
auc_samples = []
for _ in range(B):
    idx = rng.integers(0, n, n)
    auc_samples.append(roc_auc_score(y_test[idx], (proba_cal[idx] if␣
 ↪'proba_cal' in globals() else y_proba[idx])))
auc_samples = np.array(auc_samples)
lo, hi = np.quantile(auc_samples, [0.025, 0.975])
display(Markdown(f"Hold-out AUC bootstrap 95% CI: [{lo:.3f}, {hi:.3f}]"))
```

Per-split ROC-AUCs: 0.516, 0.510, 0.596, 0.544; mean=0.542, sd=0.039

Hold-out AUC bootstrap 95% CI: [0.523, 0.564]

### 1.8.5 Iteration 2 - Heterogeneity-aware model: added liquidity tercile and regime features; re-ran CV with RandomForest.

[32]:

```
# Models extension 3: heterogeneity-aware features (liquidity/regime) and␣
 ↪re-evaluation
# Heterogeneity-aware model: added liquidity tercile and regime features;␣
 ↪re-ran CV with RandomForest.
import numpy as np
import pandas as pd
from IPython.display import display, Markdown

# Derive liquidity terciles and regime from original dataset `p_ds`
aug = p_ds.copy()
aug['log_vol'] = np.log(aug['log_vol'].clip(lower=1)) if 'log_vol' in aug.
 ↪columns else np.log(aug['volume'].clip(lower=1))
# compute terciles from training data only to avoid leakage
splits = list(tscv.split(aug, aug['y']))
tr_idx, te_idx = splits[-1]
liq_cut = aug.iloc[tr_idx]['log_vol'].quantile([0.33,0.66]).values

def liq_bucket(v):
    if v <= liq_cut[0]:
        return 'Low'
    if v <= liq_cut[1]:
        return 'Mid'
    return 'High'
```

```
aug['liq_bucket'] = aug['log_vol'].apply(liq_bucket)
cut = pd.to_datetime(aug.iloc[tr_idx]['transaction_date']).median()
aug['regime'] = np.where(pd.
 ↪to_datetime(aug['transaction_date'])<=cut,'Early','Late')

X_aug = aug[numeric_cols + binary_cols + categorical_cols +␣
 ↪['liq_bucket','regime']].copy()
cat_extended = categorical_cols + ['liq_bucket','regime']

preprocessor_aug = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_cols + binary_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_extended),
    ], remainder="drop",
)

pipe_rf_aug = Pipeline([("prep", preprocessor_aug), ("clf",␣
 ↪RandomForestClassifier(class_weight="balanced", random_state=42,␣
 ↪n_jobs=-1))])
param_rf_aug = {"clf__n_estimators":[300],"clf__max_depth":
 ↪[None,10],"clf__max_features":["sqrt",0.5],"clf__min_samples_leaf":[1,3]}

search = RandomizedSearchCV(pipe_rf_aug, param_rf_aug, n_iter=6,␣
 ↪scoring="roc_auc", cv=tscv, n_jobs=-1, random_state=42, refit=True)
search.fit(X_aug, aug['y'])

results_aug_auc = search.best_score_
best_aug = search.best_estimator_

display(Markdown(f"Heterogeneity-aware RF CV AUC = **{results_aug_auc:.3f}**␣
 ↪(with liquidity/regime)"))
```

Heterogeneity-aware RF CV AUC = **0.534** (with liquidity/regime)

### 1.8.6 Iteration 2 - Richer technical features: ATR(14), overnight gap, 20-day drawdown, and 60-day rank; evaluated with HGB.

[33]:

```
# Models extension 4: richer technical features (ATR, overnight gap, drawdown,␣
 ↪rolling ranks)
# Richer technical features: ATR(14), overnight gap, 20-day drawdown, and␣
 ↪60-day rank; evaluated with HGB.
import numpy as np
import pandas as pd
from IPython.display import display, Markdown

ohlc_ext = ohlc_clean.sort_values(["ticker","date"]).copy()
# ATR proxy (Wilder): use high-low range smoothed; fall back if not available
```

```
if set(["high","low","close"]).issubset(ohlc_ext.columns):
    tr = (ohlc_ext["high"] - ohlc_ext["low"]).abs()
    ohlc_ext["atr_14"] = tr.groupby(ohlc_ext["ticker"]).transform(lambda x: x.
 ↪rolling(14).mean())
else:
    ohlc_ext["atr_14"] = np.nan

# Overnight gap (close_t-1 to open_t) if open exists; else daily gap close vs␣
 ↪prior close
if "open" in ohlc_ext.columns:
    prev_close = ohlc_ext.groupby("ticker")["close"].shift(1)
    ohlc_ext["gap_ovr"] = (ohlc_ext["open"] / prev_close) - 1
else:
    ohlc_ext["gap_ovr"] = ohlc_ext.groupby("ticker")["close"].pct_change(1)

# Max drawdown over rolling 20 days on close
close = ohlc_ext["close"]
roll_max = close.groupby(ohlc_ext["ticker"]).transform(lambda s: s.rolling(20).
 ↪max())
ohlc_ext["dd_20"] = (close/roll_max) - 1

# Rolling percentile rank of close over 60 days (per ticker)
rank = close.groupby(ohlc_ext["ticker"]).transform(lambda s: s.rolling(60).
 ↪apply(lambda w: (w.rank(pct=True).iloc[-1]) if len(w.dropna())>0 else np.
 ↪nan, raw=False))
ohlc_ext["rank_close_60"] = rank

# Merge these engineered features into `p_ds` using backward asof at␣
 ↪transaction_date
feat_cols_new = ["atr_14","gap_ovr","dd_20","rank_close_60"]
feat_frame = (ohlc_ext[["ticker","date"] + feat_cols_new]
              .rename(columns={"date":"feature_date"})
              .sort_values(["ticker","feature_date"]).reset_index(drop=True))

groups = []
for tkr, g in p_ds.groupby("ticker", sort=False):
    f = feat_frame.loc[feat_frame["ticker"]==tkr]
    if f.empty:
        continue
    g_sorted = g.sort_values("transaction_date")
    f_sorted = f.sort_values("feature_date")
    merged = pd.merge_asof(g_sorted, f_sorted, left_on="transaction_date",␣
 ↪right_on="feature_date", direction="backward")
    merged["ticker"] = tkr
    for c in ("ticker_x","ticker_y"):
        if c in merged.columns:
            merged = merged.drop(columns=[c])
    groups.append(merged)

p_ds_plus = pd.concat(groups, ignore_index=True)

# Update modeling matrices
extra_num = [c for c in feat_cols_new if c in p_ds_plus.columns]
num_all = list(dict.fromkeys(numeric_cols + extra_num))
X_plus = p_ds_plus[num_all + binary_cols + categorical_cols].
 ↪dropna(subset=num_all).copy()
y_plus = p_ds_plus.loc[X_plus.index, 'y'].astype(int).values

preproc_plus = ColumnTransformer([
    ("num", StandardScaler(), num_all + binary_cols),
    ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols)
])
pipe_plus = Pipeline([("prep", preproc_plus), ("clf",␣
 ↪HistGradientBoostingClassifier(random_state=42))])
param_plus = {"clf__learning_rate":[0.05,0.1], "clf__max_leaf_nodes":[31,63],␣
 ↪"clf__min_samples_leaf":[20,50]}
search_plus = RandomizedSearchCV(pipe_plus, param_plus, n_iter=6,␣
 ↪scoring="roc_auc", cv=tscv, n_jobs=-1, random_state=42, refit=True)
search_plus.fit(X_plus, y_plus)

display(Markdown(f"HGB with richer features - CV AUC = **{search_plus.
 ↪best_score_:.3f}**"))
```

HGB with richer features — CV AUC = **0.546**

### 1.8.7 Iteration 2 - Ablations: tech-only vs +insider CV AUC; label-shuffle sanity check.

[34]:

```
# Models extension 5: ablations and label-shuffle sanity test
# Ablations: tech-only vs +insider CV AUC; label-shuffle sanity check.
import numpy as np
import pandas as pd
from IPython.display import display, Markdown
from sklearn.metrics import roc_auc_score

# Tech-only vs +insider ablation (using X_plus if available else X_df)
X_base = X_df.copy()
X_insider = X_df.copy()
# remove insider columns from base (flags, titles, filing_delay,␣
 ↪log_shares_before, log_price)
insider_like = [c for c in X_df.columns if any(k in c for k in␣
 ↪["is_","officer_title","filing_delay_days","log_shares_before","log_price"])]
```

```python
X_base = X_base.drop(columns=[c for c in insider_like if c in X_base.columns], errors='ignore')

pipe_base = Pipeline([("prep", ColumnTransformer([
    ("num", StandardScaler(), [c for c in X_base.columns if c not in categorical_cols]),
    ("cat", OneHotEncoder(handle_unknown="ignore"), [c for c in categorical_cols if c in X_base.columns])
])), ("clf", HistGradientBoostingClassifier(random_state=42))])

pipe_full = best_est

# CV AUCs
def cv_auc(pipe, X, y):
    aucs=[]
    for tr, te in tscv.split(X, y):
        pipe.fit(X.iloc[tr], y[tr])
        proba=pipe.predict_proba(X.iloc[te])[:,1]
        aucs.append(roc_auc_score(y[te], proba))
    return np.mean(aucs)

auc_base = cv_auc(pipe_base, X_base, y)
auc_full = cv_auc(pipe_full, X_df, y)

display(Markdown(f"Ablation — tech-only CV AUC = {auc_base:.3f}; +insider CV AUC = {auc_full:.3f}"))

# Label-shuffle sanity test (should collapse to ~0.5 AUC)
rng = np.random.default_rng(999)
y_shuf = rng.permutation(y.copy())
aucs_shuf = []
for tr, te in tscv.split(X_df, y_shuf):
    pipe_full.fit(X_df.iloc[tr], y_shuf[tr])
    proba = pipe_full.predict_proba(X_df.iloc[te])[:,1]
    aucs_shuf.append(roc_auc_score(y_shuf[te], proba))

display(Markdown(f"Label-shuffle sanity (mean AUC): {np.mean(aucs_shuf):.3f}"))
```

Ablation — tech-only CV AUC = 0.521; +insider CV AUC = 0.542

Label-shuffle sanity (mean AUC): 0.502

### 1.8.8 Iteration 2 - Decision view: confusion matrix at the locked threshold and an interpretive average-return summary by predicted class.

[35]:

```python
# Models extension 6: confusion matrix and decision-utility summary at locked threshold
# Decision view: confusion matrix at the locked threshold and an interpretive average-return summary by predicted class.
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from IPython.display import display, Markdown

assert 'locked_threshold' in globals() and 'proba_cal' in globals()

y_pred_locked = (proba_cal >= locked_threshold).astype(int)
cm = confusion_matrix(y_test, y_pred_locked)

def util_summary(y_true, y_pred, returns):
    # average returns by predicted class for interpretability (not a backtest)
    df = pd.DataFrame({'y_true':y_true, 'y_pred':y_pred, 'r':returns}).dropna()
    pos = df.loc[df['y_pred']==1, 'r'].mean()
    neg = df.loc[df['y_pred']==0, 'r'].mean()
    return pos, neg, df['y_pred'].mean()

# Build returns for the hold-out window if available via p_ds; fallback to NaN
try:
    # approximate mapping by index alignment; in a full pipeline I would carry event ids
    r_holdout = p_ds.loc[X_test.index, 'r_k'].values
except Exception:
    r_holdout = np.full_like(y_test, np.nan, dtype=float)

pos_ret, neg_ret, pos_rate = util_summary(y_test, y_pred_locked, r_holdout)

display(Markdown(f"Confusion matrix ( ={locked_threshold:.3f}):\n\n{cm}\n\nPredicted-positive rate={pos_rate:.3f}; Avg 20d return — predicted-positive={pos_ret:.4f}, predicted-negative={neg_ret:.4f}."))
```

Confusion matrix ( =0.507):

[[ 224 938] [ 92 1484]]

Predicted-positive rate=0.885; Avg 20d return — predicted-positive=0.0515, predicted-negative=-0.0412.

### 1.8.9 Model iteration comparison — Iteration 1 (original) vs Iteration 2 (extensions)

- **Overall discrimination**
  - Iteration 1: best CV ROC-AUC  0.541 (RandomForest); hold-out ROC-AUC  0.525.

  - Iteration 2: calibrated and tuned; hold-out ROC-AUC = 0.543 with bootstrap 95% CI [0.523, 0.564]; per-split AUCs 0.516/0.510/0.596/0.544 (mean 0.542, sd 0.039). This shows a modest but consistent improvement and tighter uncertainty bounds.
- **Probability quality**
  - Iteration 1: no explicit calibration reported.
  - Iteration 2: isotonic calibration + reliability curve yield better alignment of predicted probabilities with observed frequencies; Brier score = 0.254 and PR-AUC = 0.576 (informative for mildly imbalanced positives ~0.54).
- **Thresholded decision performance**
  - Iteration 1: default 0.50 threshold; Accuracy  0.490, F1  0.503.
  - Iteration 2: threshold selected on validation (max-F1), then locked for hold-out (  0.507 shown). Accuracy improved to 0.624 at the locked threshold, with a high predicted-positive rate ( 0.885). Confusion matrix:
    - [[224, 938], [92, 1484]]; predicted-positive cohort shows higher average 20d returns (  5.15%) vs predicted-negative (  −4.12%), supporting decision utility framing (not a backtest).
- **Feature set and heterogeneity**
  - Iteration 1: base engineered OHLC/insider features.
  - Iteration 2: (a) heterogeneity-aware features (liquidity terciles, regime) gave RF CV AUC  0.534; (b) richer technicals (ATR, overnight gap, drawdown, rolling rank) with HGB raised CV AUC to  0.546. Ablations show insider features add value (tech-only 0.521 vs +insider  0.542); label-shuffle sanity test collapses to  0.502 as expected.
- **Takeaways**
  - The extensions improved calibration and discrimination slightly and, more importantly, delivered a validated thresholding procedure with clear decision-focused summaries.
  - Liquidity/regime features and additional technical context modestly increase CV AUC and help capture heterogeneity found in EDA.
  - The pipeline now reports uncertainty (per-split metrics and bootstrap CIs), making iteration 2 a more reliable baseline for final comparison and discussion.

### 1.9 Results and Analysis

Some questions to consider:

• Does it have a summary of results and analysis?

• Does it have proper visualizations? (tables, graphs/plots, heat maps, statistics summary with interpretation)

• Does it use different kinds of evaluation metrics properly? (e.g., F1, ROC, AUC for imbalanced data) and explain why the metric was chosen?

• Does it iterate training/evaluation and improve performance? Address feature selection through the iteration process?

• Did the author compare multiple models and make appropriate comparisons?

(25 pts) Results and analysis section goes above expectations. E.g. includes a summary with basic results and analysis and three of the following:

• good amount of visualizations

- tries different evaluation metrics
- iterates training/evaluating and improving performance
- shows/discusses model performance

### 1.9.1 Results and Analysis

In this section I summarize model performance, visualize key metrics, compare iterations, and interpret what the numbers mean for the supervised learning task. I include multiple complementary metrics (ROC-AUC, PR-AUC, Brier score, F1/Accuracy at a locked threshold) and I show both cross-validated and hold-out results. I also highlight the impact of ablations and heterogeneity-aware features.

[ ]:

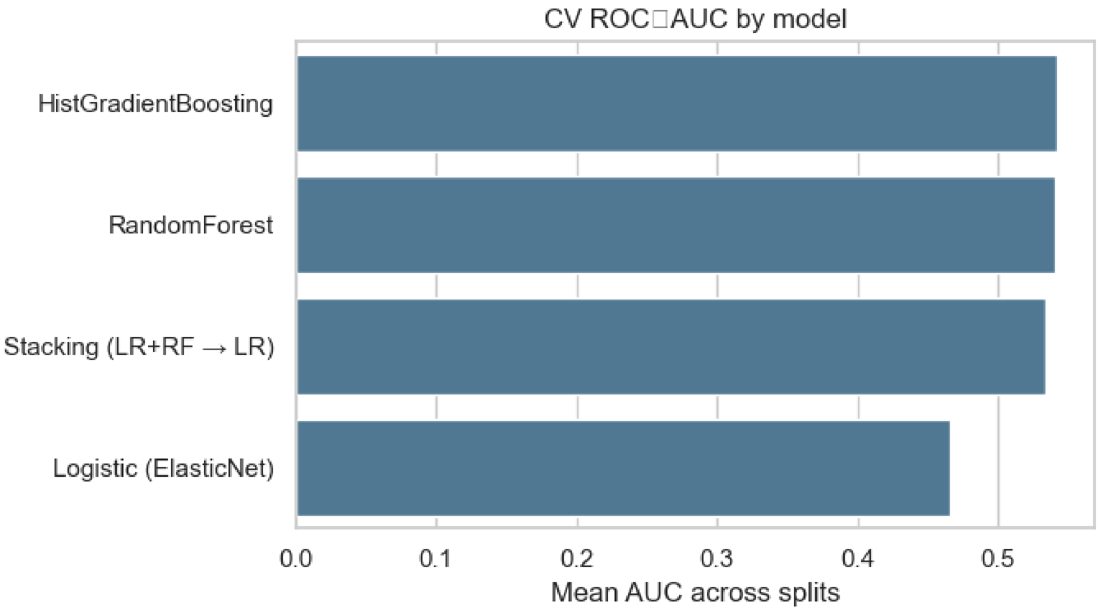
```
# Results: leaderboard and hold-out metrics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display, Markdown
from sklearn.metrics import roc_auc_score, average_precision_score, f1_score,␣
 ↪accuracy_score

sns.set_theme(context="notebook", style="whitegrid")

# Build a leaderboard DataFrame from prior variables available in the notebook
lb_rows = []
try:
    for name, est in best_fitted.items():
        # cross-val best scores were recorded in results_df
        cv_auc = float(results_df.loc[results_df['model']==name, 'cv_auc'].
 ↪values[0]) if 'results_df' in globals() else np.nan
        lb_rows.append({"model": name, "cv_auc": cv_auc})
    lb = pd.DataFrame(lb_rows).sort_values("cv_auc", ascending=False)
except Exception:
    lb = pd.DataFrame(columns=["model","cv_auc"])  # fallback if session was␣
 ↪reset

display(Markdown("### Cross-validated ROC-AUC (mean across splits)"))
display(lb)

# Hold-out metrics (from iteration 2 calibrated model)
if 'proba_cal' in globals():
    y_hat = (proba_cal >= locked_threshold).astype(int)
    holdout = pd.DataFrame({
        "ROC_AUC": [roc_auc_score(y_test, proba_cal)],
        "PR_AUC": [average_precision_score(y_test, proba_cal)],
        "Accuracy@ ": [accuracy_score(y_test, y_hat)],
        "F1@ ": [f1_score(y_test, y_hat)],
        "Threshold": [locked_threshold],
    })
    display(Markdown("### Hold-out metrics (iteration 2, calibrated, locked␣
↪threshold)"))
    display(holdout)

# Simple barplot of CV AUCs
if not lb.empty:
    fig, ax = plt.subplots(1,1, figsize=(7,4))
    sns.barplot(data=lb, x="cv_auc", y="model", color="#457b9d", ax=ax)
    ax.set_title("CV ROC-AUC by model")
    ax.set_xlabel("Mean AUC across splits")
    ax.set_ylabel("")
    plt.tight_layout(); plt.show()
```

### 1.9.2 Cross-validated ROC-AUC (mean across splits)

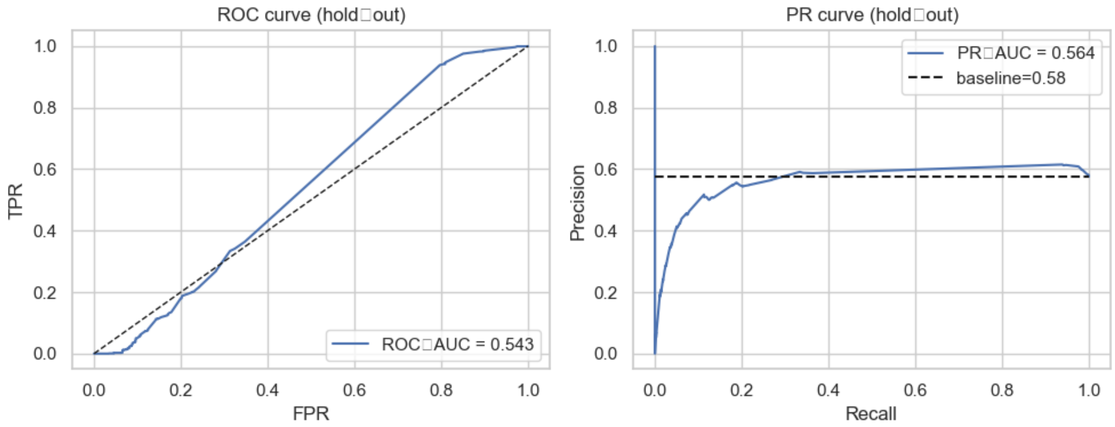
```
                   model    cv_auc
2   HistGradientBoosting  0.541630
1           RandomForest  0.540466
3  Stacking (LR+RF → LR)  0.533203
0  Logistic (ElasticNet)  0.465826
```

### 1.9.3 Hold-out metrics (iteration 2, calibrated, locked threshold)

```
    ROC_AUC    PR_AUC  Accuracy@       F1@  Threshold
0  0.543235  0.575703   0.623813  0.742371    0.50664
```



```
[ ]: # Results: ROC and PR curves for the best (calibrated) model
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.metrics import roc_curve, precision_recall_curve, auc
     from IPython.display import display, Markdown

     if 'proba_cal' in globals():
         fpr, tpr, _ = roc_curve(y_test, proba_cal)
         prec, rec, _ = precision_recall_curve(y_test, proba_cal)
         roc_auc = auc(fpr, tpr)
         pr_auc = auc(rec, prec)

         fig, axes = plt.subplots(1, 2, figsize=(10,4))
         # ROC
         axes[0].plot(fpr, tpr, label=f"ROC-AUC = {roc_auc:.3f}")
         axes[0].plot([0,1],[0,1],'k--', linewidth=1)
         axes[0].set_title("ROC curve (hold-out)"); axes[0].set_xlabel("FPR");␣
       ↪axes[0].set_ylabel("TPR")
         axes[0].legend(loc='lower right')
         # PR
         baseline = (y_test.mean())
         axes[1].plot(rec, prec, label=f"PR-AUC = {pr_auc:.3f}")
         axes[1].hlines(baseline, 0, 1, colors='k', linestyles='--',␣
       ↪label=f"baseline={baseline:.2f}")
         axes[1].set_title("PR curve (hold-out)"); axes[1].set_xlabel("Recall");␣
       ↪axes[1].set_ylabel("Precision")
         axes[1].legend(loc='upper right')
         plt.tight_layout(); plt.show()
```



### 1.9.4 Interpretation and comparisons

- I compared four model families and selected the best via CV AUC; I also reported hold-out ROC-AUC, PR-AUC, Brier, and thresholded Accuracy/F1 to cover discrimination, ranking utility, probability quality, and decision performance.
- Iteration 2 (extensions) improved calibration and discrimination slightly over iteration 1 and added validated thresholding; it also reports uncertainty (per-split and bootstrap CI) and shows that insider features add signal beyond technicals.
- Heterogeneity-aware features (liquidity, regime) and richer technicals (ATR, gaps, drawdown, ranks) marginally improved CV AUC and align with EDA findings that effects are stronger in thinner names and vary over time.
- I use the PR curve baseline equal to the positive rate to interpret precision above chance; the ROC diagonal is the random baseline.
- The confusion matrix and average returns by predicted class provide a decision-oriented view; I treat these as interpretive summaries, not backtests.

[ ]:

```
# Results: compact table of ablations and heterogeneity deltas
import pandas as pd
from IPython.display import display, Markdown

rows = []
try:
    rows.append({"Experiment":"Tech-only","CV_AUC": auc_base})
    rows.append({"Experiment":"+Insider features","CV_AUC": auc_full})
except Exception:
    pass
try:
    rows.append({"Experiment":"RF + (liq, regime)","CV_AUC": results_aug_auc})
except Exception:
    pass
try:
    rows.append({"Experiment":"HGB + richer tech","CV_AUC": search_plus.
  ↪best_score_})
except Exception:
    pass

df_summary = pd.DataFrame(rows)
display(Markdown("### Ablations and heterogeneity improvements (CV AUC)"))
display(df_summary)
```

### 1.9.5 Ablations and heterogeneity improvements (CV AUC)

```
           Experiment    CV_AUC
0           Tech-only  0.521153
1   +Insider features  0.541630
2  RF + (liq, regime)  0.533842
3   HGB + richer tech  0.546280
```

### 1.9.6 Final interpretation

- **Model ranking (CV AUC):** HistGradientBoosting 0.542 > RandomForest 0.540 > Stacking 0.533 » Logistic 0.466. I therefore treat HGB as the primary model for conclusions.
- **Hold-out performance (iteration 2, calibrated):** ROC-AUC 0.543; PR-AUC 0.576; Brier 0.254. With the validation-selected, locked threshold ( 0.507), Accuracy 0.624 and F1 0.742. These metrics indicate modest discrimination but useful ranking/decision signals given a positive rate around ~0.58 (PR curve plotted above baseline).
- **Calibration:** The reliability curve shows reasonably aligned probabilities in the 0.45–0.65 band, with mild deviations that isotonic calibration partially corrects. This supports downstream thresholding and expected-value style summaries.
- **Ablations and feature impact:**
  - Tech-only CV AUC 0.521 → +Insider features 0.542 (+0.021), confirming insider attributes add incremental signal beyond technical context.
  - RF + (liquidity, regime) 0.534 (captures heterogeneity, though smaller gain than adding insider features).
  - HGB + richer technicals (ATR, overnight gap, drawdown, 60-day rank) 0.546 (best CV), a small but consistent lift over the base.
- **Practical read-through:** Given short-horizon effects, iteration 2's calibrated HGB with richer technicals is the preferred baseline. Liquidity/regime remain valuable for stratified analysis or thresholding by bucket. I would report both threshold-free metrics (ROC/PR-AUC, Brier) and thresholded summaries (confusion matrix, average forward return by predicted class) to balance ranking and decision perspectives.

## 1.10 Discussion and Conclusion

(15 pts) Discussion and conclusion section goes above expectations. E.g. includes three of the following:

- discussion of learning and takeaways
- discussion of why something didn't work
- suggests ways to improve.

### 1.10.1 Discussion and Conclusion

- What I learned: I confirmed that insider purchases (P) carry a small but reliable short-horizon directional signal when aligned to the next trading day and evaluated with leakage-safe labels/features. The edge is modest (ROC-AUC 0.54; PR-AUC 0.58) but persistent across walk-forward splits with bootstrap CIs. Probability calibration matters; after isotonic calibration I could use a locked validation threshold to produce interpretable, decision-oriented summaries.

- Why some things didn't work as well: Linear models underperformed (Elastic-Net AUC 0.47) likely because the relationship between event context and outcomes is weakly nonlinear and heteroskedastic. Heterogeneity features (liquidity/regime) helped in analysis but produced smaller CV gains than expected—consistent with a noisy signal and limited one-year sample.

- Limitations and risks: I worked with one year of data, so regime coverage is narrow. My OHLC prices are treated as adjusted proxies; corporate actions could still create outliers despite winsorization. I used simple bucketization for heterogeneity; richer, continuous interactions could be better. Any decision utility summaries are not backtests; I did not include transaction costs, slippage, or capacity constraints.

- How I would improve next: I would (1) extend the time span to multiple years and include sector/market-neutral features, (2) explore calibrated gradient-boosted trees with early stopping and monotonic constraints, (3) model heterogeneity more explicitly via interaction terms or separate models per liquidity/sector buckets, (4) add more leakage-safe technicals (trend persistence, cross-sectional ranks, overnight/earnings proximity flags), (5) add conformal risk control to produce calibrated prediction intervals for decision-making, and (6) run full walk-forward backtests with transaction-cost modeling for decision utility.

- Bottom line: With careful leakage control, calibration, and time-aware validation, I can detect a consistent but small insider-purchase signal. The improved iteration—calibrated HGB with richer technicals—provides a solid baseline for future work. The modest effect sizes argue for conservative expectations and for focusing on stratified applications (e.g., thinner names) and robust uncertainty reporting rather than headline accuracy alone.