In [4]:
```
#Niraj Neupane

# ============================================================
# Project 7: Risk-Based Modeling (Risk Metrics + Backtesting + ML)
# ============================================================
# What it does:
# 1) Downloads prices (Stooq via direct CSV; no API keys) for GLD, QQQ, SPY, TLT
# 2) Builds an equal-weight portfolio
# 3) Plots:
#    - Adjusted prices (proxy)
#    - Portfolio equity curve (Growth of $1)
#    - Portfolio daily return distribution (histogram)
#    - Rolling volatility
#    - Drawdown
# 4) Computes risk metrics:
#    - Annualized return/vol, Sharpe
#    - Max drawdown
#    - Historical VaR & CVaR (95%)
# 5) VaR backtesting (Kupiec unconditional coverage test)
# 6) Simple ML: predict “high-risk day” (large loss) using lagged features
#
# Dependencies (safe defaults):
#   pip install pandas numpy matplotlib scikit-learn scipy
#
# Notes:
# - This version DOES NOT use pandas_datareader (avoids source/connector issues).
# - Stooq US ETFs are queried as: spy.us, qqq.us, gld.us, tlt.us
# - "Close" is used as an adjusted-price proxy for this project.
# ============================================================

import warnings
warnings.filterwarnings("ignore")

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

from scipy.stats import chi2
from sklearn.model_selection import TimeSeriesSplit
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score


# ----------------------------
# Config
# ----------------------------
TICKERS = ["GLD", "QQQ", "SPY", "TLT"]
START = "2012-01-01"
END = None  # None = up to latest available from Stooq

WEIGHTS = np.array([1 / len(TICKERS)] * len(TICKERS))  # equal weight
ALPHA = 0.05                 # 95% VaR/CVaR
```

```python
ROLL_VOL_WINDOW = 63          # ~ 3 months
ROLL_VAR_WINDOW = 252         # ~ 1 year rolling VaR
SEED = 42


# ----------------------------
# Helpers: Data
# ----------------------------
def load_stooq_close(ticker: str) -> pd.Series:
    """
    Download daily prices from Stooq via CSV and return Close as a Series.
    Stooq US tickers use {ticker}.us in lowercase.
    """
    sym = f"{ticker.lower()}.us"
    url = f"https://stooq.com/q/d/l/?s={sym}&i=d"
    df = pd.read_csv(url)

    if "Date" not in df.columns or "Close" not in df.columns:
        raise ValueError(f"Unexpected Stooq format for {ticker}. Columns: {list(df.

    df["Date"] = pd.to_datetime(df["Date"])
    df = df.sort_values("Date").set_index("Date")

    s = df["Close"].astype(float).rename(ticker.upper())
    return s


def load_prices(tickers, start=None, end=None) -> pd.DataFrame:
    series_list = [load_stooq_close(tk) for tk in tickers]
    prices = pd.concat(series_list, axis=1).dropna(how="all")

    if start is not None:
        prices = prices.loc[prices.index >= pd.to_datetime(start)]
    if end is not None:
        prices = prices.loc[prices.index <= pd.to_datetime(end)]

    # Require common dates across assets for clean portfolio math
    prices = prices.dropna(how="any")

    if prices.empty:
        raise ValueError("Prices are empty after filtering. Try a wider date range.
    return prices


# ----------------------------
# Helpers: Risk / Performance
# ----------------------------
def annualize_return(daily_returns: pd.Series) -> float:
    r = daily_returns.dropna()
    if len(r) == 0:
        return np.nan
    return float((1 + r).prod() ** (252 / len(r)) - 1)


def annualize_vol(daily_returns: pd.Series) -> float:
    r = daily_returns.dropna()
    return float(r.std(ddof=0) * np.sqrt(252))


def sharpe_ratio(daily_returns: pd.Series, rf_annual: float = 0.0) -> float:
    """
    Sharpe with constant annual RF converted to daily RF.
    """
    r = daily_returns.dropna()
    rf_daily = (1 + rf_annual) ** (1 / 252) - 1
    excess = r - rf_daily
    denom = excess.std(ddof=0)
    if denom == 0:
        return np.nan
    return float(np.sqrt(252) * excess.mean() / denom)


def max_drawdown(equity: pd.Series) -> float:
    eq = equity.dropna()
    peak = eq.cummax()
    dd = eq / peak - 1.0
    return float(dd.min())


def hist_var_cvar(daily_returns: pd.Series, alpha: float = 0.05) -> tuple[float, fl
    """
    Historical VaR and CVaR at alpha (e.g., 5%).
    Returns VaR/CVaR as positive loss numbers (e.g., 0.02 = 2% loss).
    """
    r = daily_returns.dropna()
    q = r.quantile(alpha)  # usually negative
    var = -float(q)
    tail = r[r <= q]
    cvar = -float(tail.mean()) if len(tail) > 0 else np.nan
    return var, cvar


def kupiec_uc_test(violations: int, n: int, alpha: float) -> tuple[float, float]:
    """
    Kupiec unconditional coverage test for VaR.
    Returns: (LR_uc statistic, p_value)
    """
    if n <= 0:
        return np.nan, np.nan

    pi = violations / n
    if pi <= 0 or pi >= 1:
        return np.nan, np.nan

    lr = -2 * (
        (n - violations) * np.log((1 - alpha) / (1 - pi)) +
        violations * np.log(alpha / pi)
    )
    pval = 1 - chi2.cdf(lr, df=1)
    return float(lr), float(pval)
```

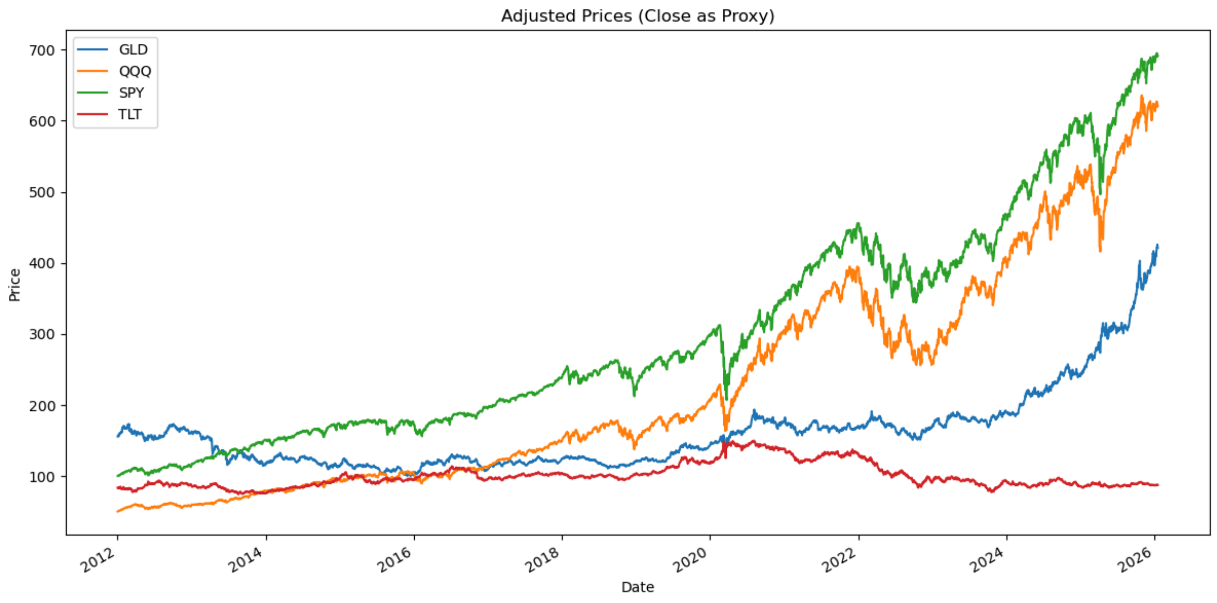
```
# ------------------------------
# 1) Load Data
# ------------------------------
print("Loading prices from Stooq...")
prices = load_prices(TICKERS, start=START, end=END)

print("\nData loaded:")
print("Tickers:", list(prices.columns))
print("Date range:", prices.index.min().date(), "to", prices.index.max().date())
print("Obs:", len(prices))


# ------------------------------
# 2) Returns + Portfolio
# ------------------------------
rets = prices.pct_change().dropna()
port_rets = rets.dot(WEIGHTS)
port_equity = (1 + port_rets).cumprod()


# ------------------------------
# 3) Plots
# ------------------------------
# (A) Prices
ax = prices.plot(figsize=(12, 6), title="Adjusted Prices (Close as Proxy)")
ax.set_xlabel("Date")
ax.set_ylabel("Price")
plt.tight_layout()
plt.show()

# (B) Portfolio Equity Curve
ax = port_equity.plot(figsize=(12, 6), title="Portfolio Equity Curve (Growth of $1)
ax.set_xlabel("Date")
ax.set_ylabel("Equity")
plt.tight_layout()
plt.show()

# (C) Return Distribution
plt.figure(figsize=(12, 6))
plt.hist(port_rets.values, bins=60)
plt.title("Portfolio Daily Return Distribution")
plt.xlabel("Return")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# (D) Rolling Volatility
roll_vol = port_rets.rolling(ROLL_VOL_WINDOW).std(ddof=0) * np.sqrt(252)
ax = roll_vol.plot(figsize=(12, 6), title=f"Rolling Volatility (Annualized, {ROLL_V
ax.set_xlabel("Date")
ax.set_ylabel("Volatility")
plt.tight_layout()
plt.show()

# (E) Drawdown
running_max = port_equity.cummax()
```

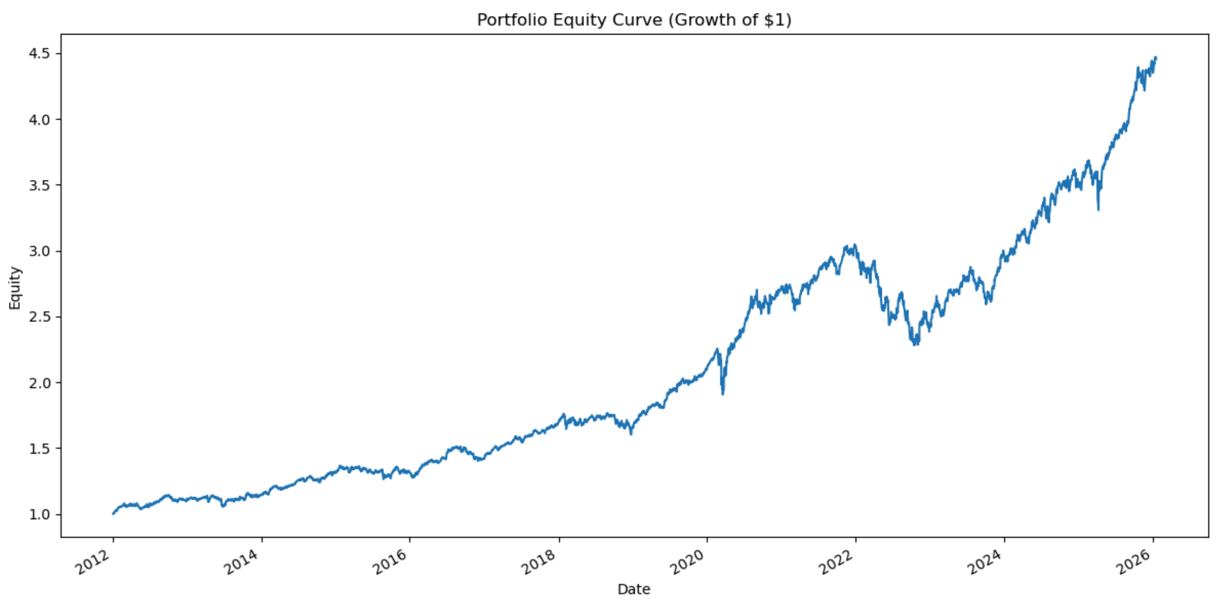
```python
drawdown = port_equity / running_max - 1
ax = drawdown.plot(figsize=(12, 6), title="Portfolio Drawdown")
ax.set_xlabel("Date")
ax.set_ylabel("Drawdown")
plt.tight_layout()
plt.show()


# ----------------------------
# 4) Risk Summary Metrics
# ----------------------------
ann_ret = annualize_return(port_rets)
ann_vol = annualize_vol(port_rets)
sharpe = sharpe_ratio(port_rets, rf_annual=0.0)
mdd = max_drawdown(port_equity)
var95, cvar95 = hist_var_cvar(port_rets, alpha=ALPHA)

summary = pd.DataFrame(
    {
        "Annualized Return": [ann_ret],
        "Annualized Volatility": [ann_vol],
        "Sharpe (rf=0)": [sharpe],
        "Max Drawdown": [mdd],
        "Hist VaR 95% (1d)": [var95],
        "Hist CVaR 95% (1d)": [cvar95],
    }
)

print("\n=== Portfolio Risk Summary ===")
print(summary.round(4))


# ----------------------------
# 5) Rolling VaR + Backtest (Kupiec UC)
# ----------------------------
rolling_var = (
    port_rets.rolling(ROLL_VAR_WINDOW)
    .quantile(ALPHA)
    .mul(-1)       # positive loss threshold
    .dropna()
)

aligned_rets = port_rets.loc[rolling_var.index]
violations = int((aligned_rets < -rolling_var).sum())  # breach if return < -VaR
n_bt = int(len(aligned_rets))

lr_uc, p_uc = kupiec_uc_test(violations, n_bt, ALPHA)

print("\n=== VaR Backtest (Kupiec UC) ===")
print(f"Window: {ROLL_VAR_WINDOW} days | Alpha: {ALPHA:.2%}")
print(f"Observations tested: {n_bt}")
print(f"Violations: {violations} (expected ~ {ALPHA*n_bt:.1f})")
print(f"Kupiec LR_uc: {lr_uc:.4f} | p-value: {p_uc:.4f}")

plt.figure(figsize=(12, 6))
aligned_rets.plot(label="Portfolio Return")
```

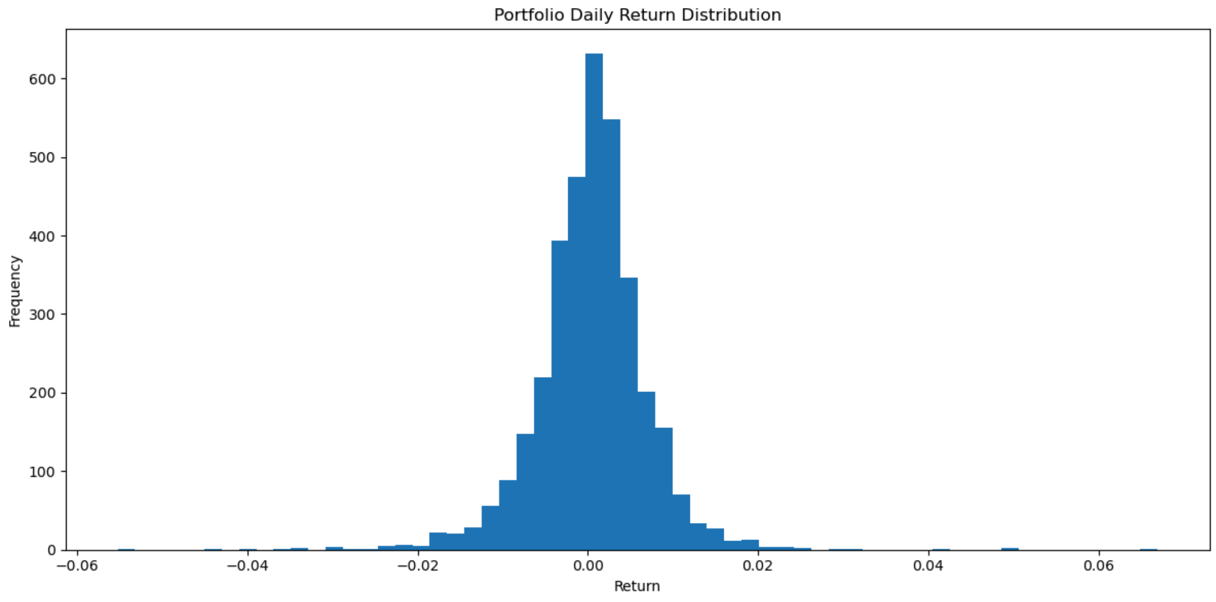
```
(-rolling_var).plot(label="VaR 95% Threshold")
plt.title("Portfolio Returns vs Rolling Historical VaR (95%)")
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.legend()
plt.tight_layout()
plt.show()


# ----------------------------
# 6) ML: Predict “High-Risk Day” (large loss)
# ----------------------------
# Target: 1 if next-day return is in worst ALPHA tail, else 0.
# Features use ONLY info available up to time t (shifted).
df = pd.DataFrame({"r": port_rets}).dropna()

df["r_lag1"] = df["r"].shift(1)
df["r_lag5"] = df["r"].rolling(5).sum().shift(1)
df["vol_20"] = df["r"].rolling(20).std(ddof=0).shift(1)
df["mom_20"] = df["r"].rolling(20).sum().shift(1)

eq = (1 + df["r"]).cumprod()
dd = eq / eq.cummax() - 1
df["dd_lag"] = dd.shift(1)

df["r_next"] = df["r"].shift(-1)
cut = df["r_next"].quantile(ALPHA)
df["y"] = (df["r_next"] <= cut).astype(int)

df = df.dropna()

feature_cols = ["r_lag1", "r_lag5", "vol_20", "mom_20", "dd_lag"]
X = df[feature_cols]
y = df["y"]

tscv = TimeSeriesSplit(n_splits=5)
pipe = Pipeline(
    steps=[
        ("scaler", StandardScaler()),
        ("clf", LogisticRegression(max_iter=2000, random_state=SEED)),
    ]
)

auc_scores = []
for train_idx, test_idx in tscv.split(X):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    pipe.fit(X_train, y_train)
    proba = pipe.predict_proba(X_test)[:, 1]
    auc_scores.append(roc_auc_score(y_test, proba))

print("\n=== ML Risk Model (Logistic Regression) ===")
print(f"Target: next-day return in worst {ALPHA:.0%} tail")
print(f"Features: {feature_cols}")
print(f"TimeSeries CV AUC (mean): {np.mean(auc_scores):.3f} | AUC (std): {np.std(au
```

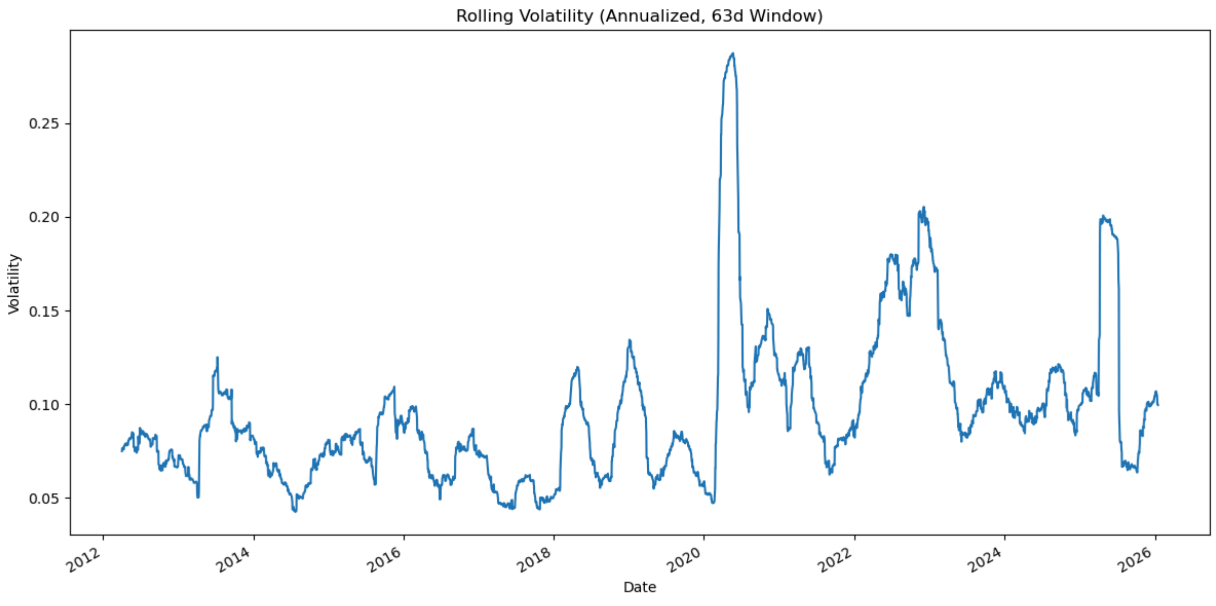
```python
# Fit on full data and plot predicted risk probability
pipe.fit(X, y)
df["risk_prob_next_day"] = pipe.predict_proba(X)[:, 1]

print("\nLast 10 days (predicted probability of a high-risk next day):")
print(df["risk_prob_next_day"].tail(10).to_frame().round(4))

plt.figure(figsize=(12, 6))
df["risk_prob_next_day"].plot(title="ML Risk Forecast: P(Next Day is High-Risk)")
plt.xlabel("Date")
plt.ylabel("Probability")
plt.tight_layout()
plt.show()
```

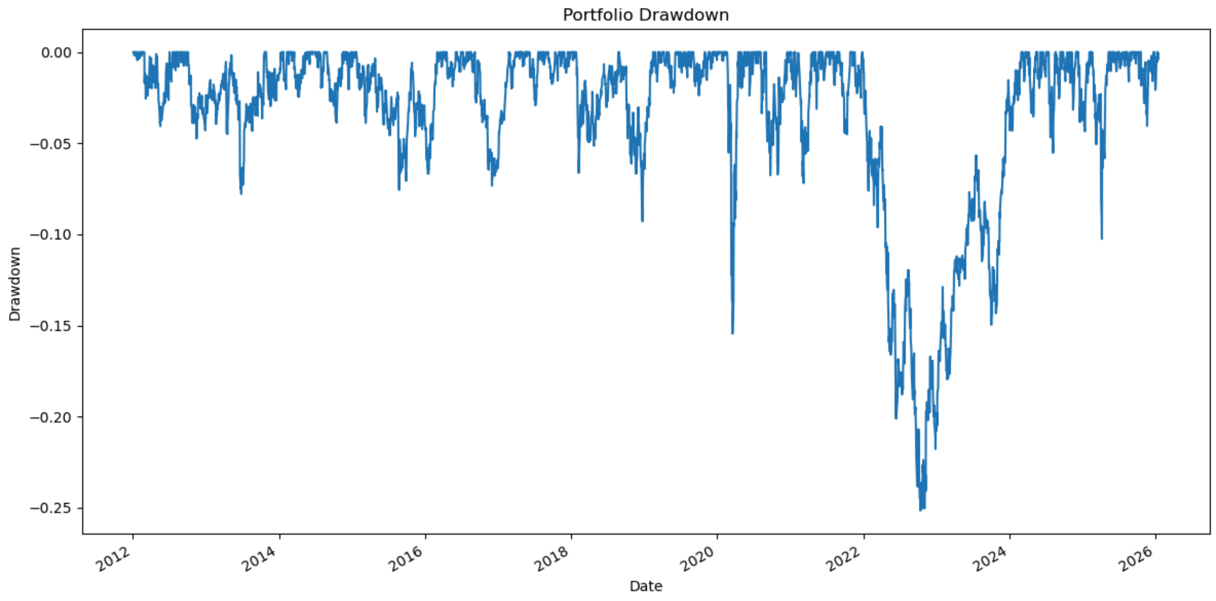
```
Loading prices from Stooq...

Data loaded:
Tickers: ['GLD', 'QQQ', 'SPY', 'TLT']
Date range: 2012-01-03 to 2026-01-16
Obs: 3531
```



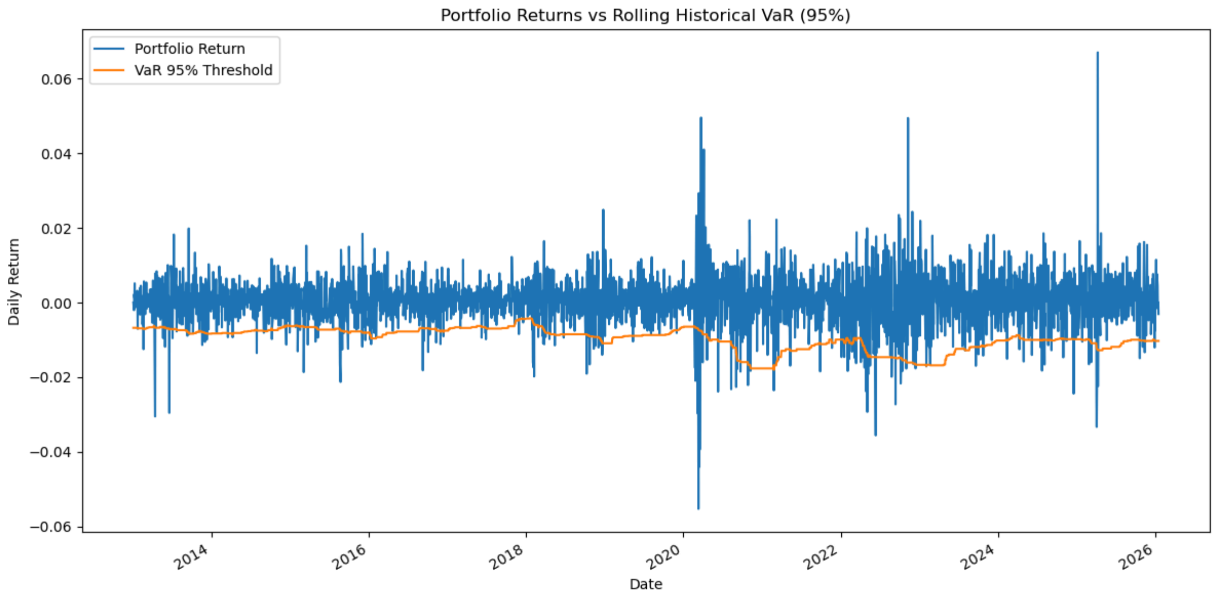

Portfolio Equity Curve (Growth of $1)
Equity
Date
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
2012
2014
2016
2018
2020
2022
2024
2026


Portfolio Daily Return Distribution
Frequency
Return
0
100
200
300
400
500
600
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06

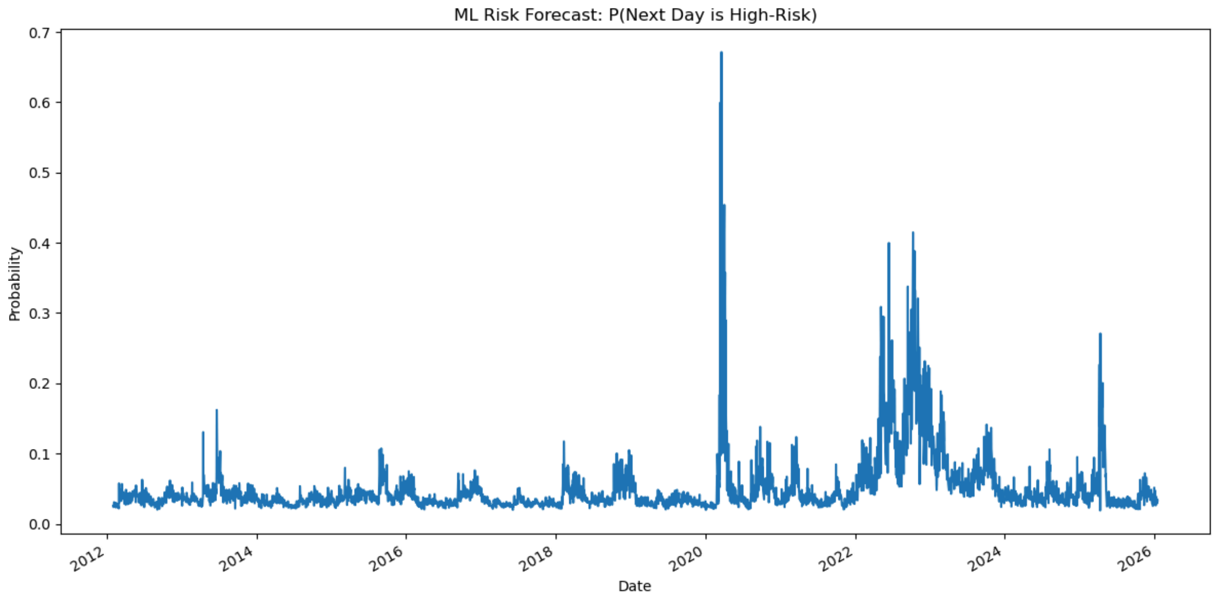

Rolling Volatility (Annualized, 63d Window)
Volatility
Date
0.05
0.10
0.15
0.20
0.25
2012
2014
2016
2018
2020
2022
2024
2026



```
=== Portfolio Risk Summary ===
   Annualized Return  Annualized Volatility  Sharpe (rf=0)  Max Drawdown  \
0             0.1125                 0.1044         1.0734       -0.2516

   Hist VaR 95% (1d)  Hist CVaR 95% (1d)
0             0.0098              0.0153

=== VaR Backtest (Kupiec UC) ===
Window: 252 days | Alpha: 5.00%
Observations tested: 3279
Violations: 167 (expected ~ 164.0)
Kupiec LR_uc: 0.0594 | p-value: 0.8075
```



```
=== ML Risk Model (Logistic Regression) ===
Target: next-day return in worst 5% tail
Features: ['r_lag1', 'r_lag5', 'vol_20', 'mom_20', 'dd_lag']
TimeSeries CV AUC (mean): 0.565 | AUC (std): 0.087

Last 10 days (predicted probability of a high-risk next day):
            risk_prob_next_day
Date
2026-01-02              0.0477
2026-01-05              0.0373
2026-01-06              0.0262
2026-01-07              0.0303
2026-01-08              0.0377
2026-01-09              0.0384
2026-01-12              0.0290
2026-01-13              0.0313
2026-01-14              0.0345
2026-01-15              0.0334
```


ML Risk Forecast: P(Next Day is High-Risk)
Probability
Date