

In [1]:

```
# =====
# Project 4: Options Strategy & Volatility Modeling (SPY + VIX proxy)
# =====
# What this notebook/script does:
# 1) Downloads SPY prices and VIX index data (as an implied vol proxy)
# 2) Computes realized volatility (RV) from SPY returns
# 3) Compares implied vol proxy (IV ~ VIX/100) vs realized vol
# 4) Implements Black-Scholes pricing + Greeks (Delta, Gamma, Vega, Theta)
# 5) Runs a simple 30D ATM Straddle backtest:
#     - Buy ATM call + put (priced with IV proxy) at entry
#     - Hold ~30 calendar days (~21 trading days)
#     - Settle at intrinsic value at exit date
# 6) Saves outputs to ./outputs and plots key charts
#
# Requirements:
#   pip install numpy pandas matplotlib yfinance
# =====

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

try:
    import yfinance as yf
except ModuleNotFoundError as e:
    raise ModuleNotFoundError("Missing yfinance. Install with: pip install yfinance")

plt.style.use("seaborn-v0_8")

# -----
# Config
# -----
OUTPUT_DIR = "outputs"
os.makedirs(OUTPUT_DIR, exist_ok=True)

UNDERLYING = "SPY"
IV_PROXY = "^VIX"           # VIX as proxy for SPY implied vol (approx)
START = "2010-01-01"
END = None

TRADING_DAYS_PER_YEAR = 252
RV_WINDOW = 20              # 20 trading days (~1 month) realized vol
HOLD_DAYS = 21                # ~30 calendar days in trading days
T = 30 / 365.0               # 30 calendar days to expiry
R = 0.03                     # risk-free rate (simple constant; ok for project)
Q = 0.0                       # dividend yield (simplified; SPY has divs, but ok for r

LEVERAGE = 1.0                # scale straddle exposure if you want
MIN_IV = 0.05                  # floor to avoid weird edge cases
MAX_IV = 2.00                  # cap to avoid crazy spikes
```

```

# -----
# Math helpers: Normal PDF/CDF (no scipy)
# -----
def norm_pdf(x: np.ndarray) -> np.ndarray:
    return (1.0 / np.sqrt(2 * np.pi)) * np.exp(-0.5 * x * x)

def norm_cdf(x: np.ndarray) -> np.ndarray:
    #  $\Phi(x)$  using erf
    return 0.5 * (1.0 + np.vectorize(math.erf)(x / np.sqrt(2.0)))

# -----
# Black-Scholes (European) pricing + Greeks
# -----
def bs_d1(S, K, T, r, q, sigma):
    S = np.asarray(S, dtype=float)
    K = np.asarray(K, dtype=float)
    sigma = np.asarray(sigma, dtype=float)

    return (np.log(S / K) + (r - q + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))

def bs_d2(d1, sigma, T):
    return d1 - sigma * np.sqrt(T)

def bs_call_price(S, K, T, r, q, sigma):
    d1 = bs_d1(S, K, T, r, q, sigma)
    d2 = bs_d2(d1, sigma, T)
    return S * np.exp(-q * T) * norm_cdf(d1) - K * np.exp(-r * T) * norm_cdf(d2)

def bs_put_price(S, K, T, r, q, sigma):
    d1 = bs_d1(S, K, T, r, q, sigma)
    d2 = bs_d2(d1, sigma, T)
    return K * np.exp(-r * T) * norm_cdf(-d2) - S * np.exp(-q * T) * norm_cdf(-d1)

def bs_greeks(S, K, T, r, q, sigma):
    """
    Returns a DataFrame-like dict of Greeks for call and put.
    Delta, Gamma, Vega, Theta (per day), using BS European model.
    """

    Notes:
    - Vega is per 1.00 change in vol (e.g., 0.01 vol change => vega*0.01)
    - Theta returned per DAY (divide annual theta by 365)
    """
    S = np.asarray(S, dtype=float)
    K = np.asarray(K, dtype=float)
    sigma = np.asarray(sigma, dtype=float)

    d1 = bs_d1(S, K, T, r, q, sigma)
    d2 = bs_d2(d1, sigma, T)

    pdf_d1 = norm_pdf(d1)

    # Common terms
    disc_q = np.exp(-q * T)
    disc_r = np.exp(-r * T)
    sqrtT = np.sqrt(T)

```

```

# Delta
call_delta = disc_q * norm_cdf(d1)
put_delta = disc_q * (norm_cdf(d1) - 1)

# Gamma (same for call & put)
gamma = (disc_q * pdf_d1) / (S * sigma * sqrtT)

# Vega (same for call & put)
vega = S * disc_q * pdf_d1 * sqrtT

# Theta (annual), then convert to per-day
call_theta_annual =
    - (S * disc_q * pdf_d1 * sigma) / (2 * sqrtT)
    - r * K * disc_r * norm_cdf(d2)
    + q * S * disc_q * norm_cdf(d1)
)
put_theta_annual =
    - (S * disc_q * pdf_d1 * sigma) / (2 * sqrtT)
    + r * K * disc_r * norm_cdf(-d2)
    - q * S * disc_q * norm_cdf(-d1)
)

call_theta = call_theta_annual / 365.0
put_theta = put_theta_annual / 365.0

return {
    "call_delta": call_delta,
    "put_delta": put_delta,
    "gamma": gamma,
    "vega": vega,
    "call_theta_per_day": call_theta,
    "put_theta_per_day": put_theta,
}

# -----
# Data download
# -----
def download_close(ticker: str, start: str, end=None) -> pd.Series:
    data = yf.download(ticker, start=start, end=end, auto_adjust=True, progress=False)
    if isinstance(data, pd.DataFrame) and "Close" in data.columns:
        s = data["Close"].copy()
    else:
        # fallback
        s = data.copy()
    s = s.dropna()
    s.name = ticker
    return s

print("Downloading SPY + VIX...")
spy = download_close(UNDERLYING, START, END)
vix = download_close(IV_PROXY, START, END)

df = pd.concat([spy, vix], axis=1).dropna()
df.columns = ["SPY", "VIX"]

```

```

print(df.head())

# -----
# Realized Volatility (RV)
# -----
df["ret"] = df["SPY"].pct_change()
df = df.dropna()

# RV: rolling std of daily returns, annualized
df["rv_20d"] = df["ret"].rolling(RV_WINDOW).std() * np.sqrt(TRADING_DAYS_PER_YEAR)

# IV proxy: VIX is roughly 30-day implied vol in %, convert to decimal
df["iv_proxy"] = (df["VIX"] / 100.0).clip(lower=MIN_IV, upper=MAX_IV)

# Vol risk premium proxy
df["vrp"] = df["iv_proxy"] - df["rv_20d"]

# -----
# Options (ATM) pricing + Greeks time series
# -----
# ATM: K = S at each date
df["K"] = df["SPY"]

# Prices
df["call_px"] = bs_call_price(df["SPY"].values, df["K"].values, T, R, Q, df["iv_proxy"])
df["put_px"] = bs_put_price(df["SPY"].values, df["K"].values, T, R, Q, df["iv_proxy"])
df["straddle_premium"] = df["call_px"] + df["put_px"]

# Greeks
g = bs_greeks(df["SPY"].values, df["K"].values, T, R, Q, df["iv_proxy"].values)
df["call_delta"] = g["call_delta"]
df["put_delta"] = g["put_delta"]
df["gamma"] = g["gamma"]
df["vega"] = g["vega"]
df["call_theta_per_day"] = g["call_theta_per_day"]
df["put_theta_per_day"] = g["put_theta_per_day"]
df["straddle_theta_per_day"] = df["call_theta_per_day"] + df["put_theta_per_day"]

# -----
# Strategy Backtest: 30D ATM Long Straddle (simplified, intrinsic settlement)
# -----
# Entry: at time t, buy ATM straddle priced by IV proxy
# Exit: at time t+HOLD_DAYS, settle at intrinsic value using  $S_{t+H}$ 
# PnL per straddle = intrinsic_exit - premium_entry
# intrinsic straddle =  $|S_T - K|$ , with K = S_entry
#
# Important: This is a simplified model (European ATM with 30D maturity),
# but it's great for demonstrating vol intuition and VRP dynamics.

df["S_entry"] = df["SPY"]
df["K_entry"] = df["S_entry"]
df["premium_entry"] = df["straddle_premium"]

```

```

df["S_exit"] = df["SPY"].shift(-HOLD_DAYS)

df["intrinsic_exit"] = (df["S_exit"] - df["K_entry"]).abs()
df["pnl"] = LEVERAGE * (df["intrinsic_exit"] - df["premium_entry"])
df["pnl_pct_of_spy"] = df["pnl"] / df["SPY"] # normalize by underlying level (roug

# Drop rows without exit
bt = df.dropna(subset=["S_exit", "pnl"]).copy()

# Strategy "returns" series (approx): pnl / premium (capital at risk proxy)
# Note: options require margin, but this is a common research normalization.
bt["ret_on_premium"] = bt["pnl"] / bt["premium_entry"]

# Equity curve on normalized returns
bt["equity"] = (1.0 + bt["ret_on_premium"].fillna(0.0)).cumprod()

# Drawdown
bt["equity_peak"] = bt["equity"].cummax()
bt["drawdown"] = bt["equity"] / bt["equity_peak"] - 1.0

# -----
# Metrics
# -----
def summary_stats(returns: pd.Series) -> dict:
    r = returns.dropna()
    if len(r) < 10:
        return {"n": int(len(r))}

    ann_ret = (1 + r).prod() ** (TRADING_DAYS_PER_YEAR / len(r)) - 1
    ann_vol = r.std() * np.sqrt(TRADING_DAYS_PER_YEAR)
    sharpe = np.nan if ann_vol == 0 else ann_ret / ann_vol
    win_rate = (r > 0).mean()

    return {
        "n": int(len(r)),
        "ann_return": float(ann_ret),
        "ann_vol": float(ann_vol),
        "sharpe": float(sharpe) if np.isfinite(sharpe) else np.nan,
        "win_rate": float(win_rate),
    }

metrics = summary_stats(bt["ret_on_premium"])
max_dd = float(bt["drawdown"].min())

metrics_df = pd.DataFrame([
    **metrics,
    "max_drawdown": max_dd,
    "avg_vrp": float(bt["vrp"].mean()),
    "vrp_pos_frac": float((bt["vrp"] > 0).mean()),
])
print("\n===== Strategy Metrics (Long 30D ATM Straddle) =====")
print(metrics_df.to_string(index=False))

# -----
# Save outputs

```

```

# -----
ivrv_out = df[["SPY", "VIX", "rv_20d", "iv_proxy", "vrp"]].dropna().copy()
ivrv_out.to_csv(os.path.join(OUTPUT_DIR, "options_iv_vs_rv.csv"))

bt_out = bt[[
    "SPY", "S_exit", "iv_proxy", "rv_20d", "vrp",
    "premium_entry", "intrinsic_exit", "pnl", "ret_on_premium",
    "equity", "drawdown"
]].copy()
bt_out.to_csv(os.path.join(OUTPUT_DIR, "options_straddle_backtest.csv"))

metrics_df.to_csv(os.path.join(OUTPUT_DIR, "options_strategy_metrics.csv"), index=False)

print("\nSaved outputs to ./outputs:")
print("- options_iv_vs_rv.csv")
print("- options_straddle_backtest.csv")
print("- options_strategy_metrics.csv")

# -----
# Plots
# -----
# 1) IV vs RV
plt.figure(figsize=(11, 6))
ivrv_out["iv_proxy"].plot(label="IV proxy (VIX/100)")
ivrv_out["rv_20d"].plot(label="RV 20d (ann.)", alpha=0.9)
plt.title("Implied vs Realized Volatility (SPY) - IV proxy from VIX")
plt.ylabel("Vol (decimal)")
plt.legend()
plt.tight_layout()
plt.show()

# 2) VRP
plt.figure(figsize=(11, 4))
ivrv_out["vrp"].plot()
plt.axhline(0, linewidth=1)
plt.title("Volatility Risk Premium Proxy (IV - RV)")
plt.ylabel("Vol premium (decimal)")
plt.tight_layout()
plt.show()

# 3) Straddle equity curve
plt.figure(figsize=(11, 6))
bt["equity"].plot()
plt.title("Long 30D ATM Straddle - Equity Curve (Returns on Premium)")
plt.ylabel("Equity (normalized)")
plt.tight_layout()
plt.show()

# 4) Drawdown
plt.figure(figsize=(11, 4))
bt["drawdown"].plot()
plt.title("Long 30D ATM Straddle - Drawdown")
plt.ylabel("Drawdown")
plt.tight_layout()
plt.show()

```

```

# 5) Greeks snapshot (optional)
plt.figure(figsize=(11, 6))
df[["vega", "gamma", "straddle_theta_per_day"]].dropna().tail(1500).plot()
plt.title("Greeks (ATM, 30D) - Vega / Gamma / Theta")
plt.ylabel("Greek value")
plt.tight_layout()
plt.show()

```

Downloading SPY + VIX...

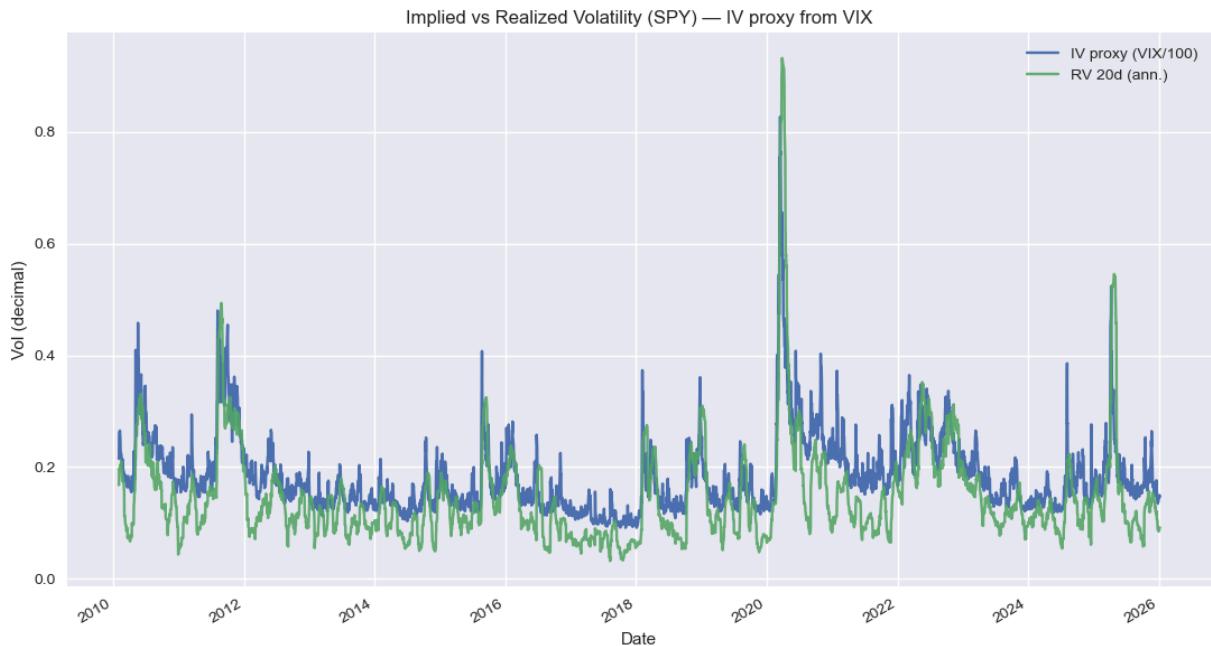
	SPY	VIX
Date		
2010-01-04	85.027931	20.040001
2010-01-05	85.253044	19.350000
2010-01-06	85.313049	19.160000
2010-01-07	85.673210	19.059999
2010-01-08	85.958328	18.129999

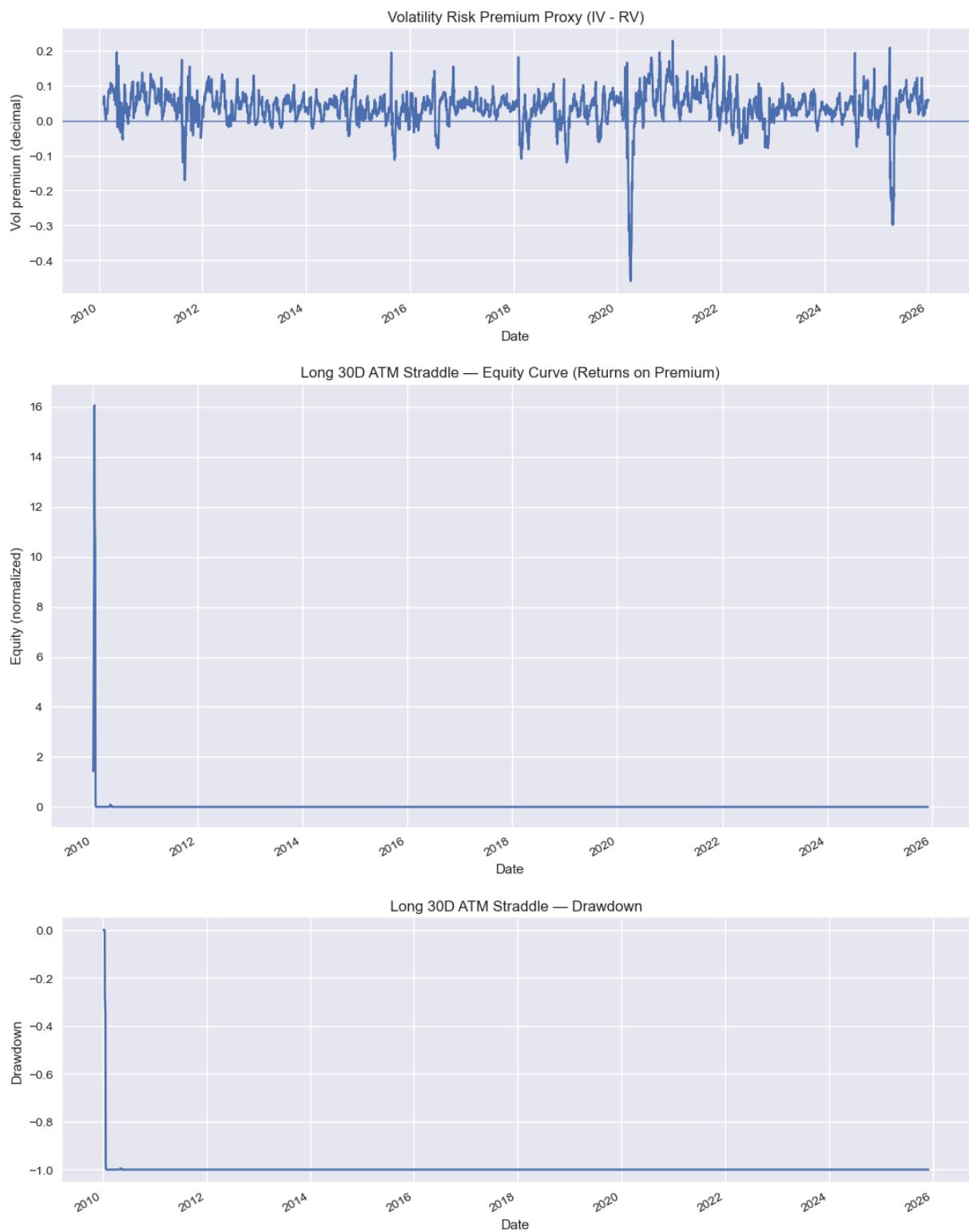
===== Strategy Metrics (Long 30D ATM Straddle) =====

n	ann_return	ann_vol	sharpe	win_rate	max_drawdown	avg_vrp	vrp_pos_frac	
4004	-1.0	10.213839	-0.097906	0.316184		-1.0	0.037019	0.849401

Saved outputs to ./outputs:

- options_iv_vs_rv.csv
- options_straddle_backtest.csv
- options_strategy_metrics.csv





<Figure size 1100x600 with 0 Axes>

