# Code HW 1.py

```python
"""
FE-621 Hw 1 – Julius Stiemer

Includes Part 1 data gathering
Includes Part 2 data gathering (TSLA, SPY, ^VIX)
"""




import argparse
import os
from datetime import datetime, timedelta
#Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm            #scipy needed for norm.cdf and pdf
import yfinance as yf




# CORE MATH BS,IV,GREEKS

class BS:                               #Black-Scholes pricing Formula
    @staticmethod
    def d1(S, K, T, r, sigma):
        return (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))

    @staticmethod
    def d2(S, K, T, r, sigma):
        return BS.d1(S, K, T, r, sigma) - sigma * np.sqrt(T)

    @staticmethod                       #standard call Formula BS
    def call(S, K, T, r, sigma):
        d1 = BS.d1(S, K, T, r, sigma)
        d2 = BS.d2(S, K, T, r, sigma)
        return float(S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2))

    @staticmethod                      #Put for BS via put-call parity
    def put(S, K, T, r, sigma):
        d1 = BS.d1(S, K, T, r, sigma)
        d2 = BS.d2(S, K, T, r, sigma)
        return float(K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1))

    @staticmethod                       #Vega (dC/dsigma)
    def vega(S, K, T, r, sigma):
        d1 = BS.d1(S, K, T, r, sigma)
        return float(S * np.sqrt(T) * norm.pdf(d1))



                                        #bisection and newton methods
class IVSolver:
    def __init__(self, tol=1e-6, max_iter=100):
        self.tol = tol
        self.max_iter = max_iter
                                            #search between low and high
    def bisection(self, market_price, S, K, T, r, kind="call"):
        lo, hi = 0.01, 5.0
        it = 0
        for _ in range(self.max_iter):
            it += 1
            mid = 0.5 * (lo + hi)
            price = BS.call(S, K, T, r, mid) if kind == "call" else BS.put(S, K, T, r, mid)
            if abs(price - market_price) < self.tol:
                return mid, it
            if price > market_price:
                hi = mid
            else:
                lo = mid
        return 0.5 * (lo + hi), it

    def newton(self, market_price, S, K, T, r, kind="call"):
        sigma = 0.5
        it = 0
        for _ in range(self.max_iter):
            it += 1
            price = BS.call(S, K, T, r, sigma) if kind == "call" else BS.put(S, K, T, r, sigma)
            diff = price - market_price
            if abs(diff) < self.tol:
                return sigma, it
                                                #avoid negative or extrem vol.
            v = BS.vega(S, K, T, r, sigma)
            if abs(v) < 1e-10:
                # if vega is ~ 0 then newton unstable
                return sigma, it

            sigma_new = sigma - diff / v
            sigma_new = max(0.001, min(5.0, float(sigma_new)))

            if abs(sigma_new - sigma) < self.tol:
                return sigma_new, it

            sigma = sigma_new

        return sigma, it

                                    # numerical and analytical greeks
class Greeks:
    @staticmethod
    def delta(S, K, T, r, sigma, kind="call"):

        d1 = BS.d1(S, K, T, r, sigma)
        return float(norm.cdf(d1) if kind == "call" else norm.cdf(d1) - 1.0)

    @staticmethod
    def gamma(S, K, T, r, sigma):                       # gamme the same for puts and calls

        d1 = BS.d1(S, K, T, r, sigma)
        return float(norm.pdf(d1) / (S * sigma * np.sqrt(T)))

    @staticmethod
    def vega(S, K, T, r, sigma):
        return BS.vega(S, K, T, r, sigma)

    @staticmethod
    def delta_cd(S, K, T, r, sigma, kind ="call", h = 0.01):
        up = BS.call(S + h, K, T, r, sigma) if kind == "call" else BS.put(S + h, K, T, r, sigma)
        dn = BS.call(S - h, K, T, r, sigma) if kind == "call" else BS.put(S - h, K, T, r, sigma)

        return float((up - dn) / (2.0 * h))

    @staticmethod
    def gamma_cd(S, K, T, r, sigma, kind = "call", h = 0.01):
        up = BS.call(S + h, K, T, r, sigma) if kind == "call" else BS.put(S + h, K, T, r, sigma)
        mid = BS.call(S, K, T, r, sigma) if kind == "call" else BS.put(S, K, T, r, sigma)
        dn = BS.call(S - h, K, T, r, sigma) if kind == "call" else BS.put(S - h, K, T, r, sigma)

        return float((up - 2.0 * mid + dn) / (h**2))

    @staticmethod
    def vega_cd(S, K, T, r, sigma, kind = "call", h = 0.01):
        up = BS.call(S, K, T, r, sigma + h) if kind == "call" else BS.put(S, K, T, r, sigma + h)
        dn = BS.call(S, K, T, r, sigma - h) if kind == "call" else BS.put(S, K, T, r, sigma - h)

        return float((up - dn) / (2.0 * h))

                        # convert date to years
def time_to_maturity(expiry_yyyy_mm_dd, asof_yyyy_mm_dd):
    exp = datetime.strptime(expiry_yyyy_mm_dd, "%Y-%m-%d")
    asof = datetime.strptime(asof_yyyy_mm_dd, "%Y-%m-%d")

    return (exp - asof).days / 365.0


def bucket_moneyness(S, K, atm_lo = 0.95, atm_hi = 1.05):
    m = S / K
    if atm_lo <= m <= atm_hi:
        return "ATM", m
    if m > atm_hi:
        return "ITM_CALL", m
    return "OTM_CALL", m

                                # put call parity (P=C-S+K*e^(-rT))
def parity_put_from_call(call_mid, S, K, T, r):
    return float(call_mid - S + K * np.exp(-r * T))


# DATA PULL (Yahoo)

def third_fridays_from(start_date, months=3):
    out = []
    for i in range(months):
        m = start_date.month + i
        y = start_date.year                     #year rollover
        while m > 12:
            m -= 12
            y += 1

        first = datetime(y, m, 1)               #find 1st Friday (+3rd Friday 2 weeks later)
        # weekdays Mon=0 -> Fri=4
        days_to_friday = (4 - first.weekday()) % 7
        first_friday = first + timedelta(days=days_to_friday)
        third_friday = first_friday + timedelta(weeks=2)
        out.append(third_friday.strftime("%Y-%m-%d"))

    return out


def pick_expiries_like_assignment(ticker, asof_date_str, how_many=3):


                                                    #match spec for expiries
    asof_dt = datetime.strptime(asof_date_str, "%Y-%m-%d")
    available = list(getattr(ticker, "options", []) or [])
    if not available:
        return []

    wanted = third_fridays_from(asof_dt, months=how_many)
    available_set = set(available)

    used = [d for d in wanted if d in available_set]
    if len(used) < how_many:
        extras = [d for d in sorted(available) if d >= asof_date_str and d not in used]
        used = (used + extras)[:how_many]
    return used


def clean_chain(df):

                                #fill missing data with next available
    if df is None or df.empty:
        return df

    df = df.copy()
    keep = [
        "contractSymbol", "strike", "bid", "ask", "lastPrice", "volume", "openInterest",
        "impliedVolatility", "inTheMoney"
    ]
    keep = [c for c in keep if c in df.columns]
    if keep:
        df = df[keep]

    for col in ["strike", "bid", "ask", "lastPrice", "volume", "openInterest"]:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors="coerce")

    # drop duplicates for contractSymbol if present
    if "contractSymbol" in df.columns:
        df = df.drop_duplicates(subset=["contractSymbol"], keep="first")

    return df


def get_data1_data2(symbols, expiries_to_use, exports_dir, quiet=False):

    print("\n" + "=" * 80)
    print("[PART 1] DATA GATHERING")
    print("=" * 80)
    print("Using last 2 trading days from the most recent 5 daily bars.\n")

    os.makedirs(exports_dir, exist_ok=True)

    data1, data2 = {}, {}
                        #main data download
    for sym in symbols:
        if not quiet:
            print(f"Symbol: {sym}")

        try:
            t = yf.Ticker(sym)
            hist = t.history(period="5d", interval="1d")

            if hist is None or hist.empty or len(hist) < 2:
                raise ValueError("not enough daily bars")

            date1 = hist.index[-2].strftime("%Y-%m-%d")
            date2 = hist.index[-1].strftime("%Y-%m-%d")
            px1 = float(hist["Close"].iloc[-2])
            px2 = float(hist["Close"].iloc[-1])

            if not quiet:
                print(f"  DATA1: {date1} close = ${px1:.2f}")
                print(f"  DATA2: {date2} close = ${px2:.2f}")

            hist.to_csv(os.path.join(exports_dir, f"history_{sym.replace('^','')}.csv"))

            if sym == "^VIX":
                data1[sym] = {"price": px1, "date": date1, "history": hist}
                data2[sym] = {"price": px2, "date": date2, "history": hist}
                if not quiet:
                    print("")
                continue

            expiries = pick_expiries_like_assignment(t, date1, how_many=expiries_to_use)
            if not expiries:
                raise ValueError("no option expiries found")

            if not quiet:
                print("  expiries used:", expiries)

            calls_list, puts_list = [], []
            for exp in expiries:
                chain = t.option_chain(exp)
                calls_raw = chain.calls.copy()
                puts_raw = chain.puts.copy()

                base = f"{sym}_{exp}".replace("^", "")
                calls_raw.to_csv(os.path.join(exports_dir, f"raw_calls_{base}.csv"), index=False)
                puts_raw.to_csv(os.path.join(exports_dir, f"raw_puts_{base}.csv"), index=False)

                calls = clean_chain(calls_raw)
                puts = clean_chain(puts_raw)

                calls["expiry"] = exp
                puts["expiry"] = exp

                calls.to_csv(os.path.join(exports_dir, f"calls_{base}.csv"), index=False)
                puts.to_csv(os.path.join(exports_dir, f"puts_{base}.csv"), index=False)

                calls_list.append(calls)
                puts_list.append(puts)

            data1[sym] = {"price": px1, "date": date1, "history": hist, "calls": calls_list, "puts": puts_list, "expiries": expiries
            data2[sym] = {"price": px2, "date": date2, "history": hist, "calls": calls_list, "puts": puts_list, "expiries": expiries

            if not quiet:
                print("")

        except Exception as e:
            print(f"  [warn] {sym} download failed: {e}\n")

    # spot summary CSV
    rows = []
    for s in symbols:
        if s in data1:
            rows.append({"dataset": "DATA1", "symbol": s, "date": data1[s]["date"], "price": data1[s]["price"]})
        if s in data2:
            rows.append({"dataset": "DATA2", "symbol": s, "date": data2[s]["date"], "price": data2[s]["price"]})
    if rows:
        pd.DataFrame(rows).to_csv(os.path.join(exports_dir, "spot_prices_DATA1_DATA2.csv"), index=False)

    print("\n" + "=" * 80)
    print("SYMBOL DESCRIPTIONS (Problem 3)")
    print("=" * 80)
    print("TSLA: Tesla common stock (NASDAQ).")
    print("SPY: S&P 500 ETF (tracks broad US equity index).")
    print("^VIX: volatility index implied by SPX options (market 'fear gauge').")
    print("Options have many expiries (weeklies/monthlies) for different hedging/trading horizons.\n")

    return data1, data2


# REAL OPTIONS TASKS

def valid_call_rows(calls_df):


    #  bid > 0 and ask > 0 and (volume > 0 OR openInterest > 0)

    df = calls_df.copy()

    if "bid" not in df.columns or "ask" not in df.columns:
        return df.iloc[0:0].copy()

    if "volume" in df.columns:
        df["volume"] = df["volume"].fillna(0)
    else:
        df["volume"] = 0

    if "openInterest" in df.columns:
        df["openInterest"] = df["openInterest"].fillna(0)
    else:
        df["openInterest"] = 0

    df = df[(df["bid"] > 0) & (df["ask"] > 0) & ((df["volume"] > 0) | (df["openInterest"] > 0))].copy()
    return df


def iv_table_problems4_8(data1, symbol, r1, exports_dir, atm_lo=0.95, atm_hi=1.05):

    print("\n" + "=" * 80)
    print(f"[PART 2] IV TABLES (DATA1) — {symbol}")
    print("=" * 80)

    if symbol not in data1 or "calls" not in data1[symbol]:
        print(f"[warn] no calls for {symbol}")
        return None

    S = data1[symbol]["price"]
    asof = data1[symbol]["date"]
    expiries = data1[symbol]["expiries"]
    solver = IVSolver()

    rows = []
    for calls_df, exp in zip(data1[symbol]["calls"], expiries):
        T = time_to_maturity(exp, asof)
        vdf = valid_call_rows(calls_df)
        print(f"  expiry {exp} (T={T*365:.0f}d) valid={len(vdf)}")

        for _, opt in vdf.iterrows():
            K = float(opt["strike"])
            mid = float((opt["bid"] + opt["ask"]) / 2.0)
            cls, m = bucket_moneyness(S, K, atm_lo, atm_hi)

            try:
                iv_b, it_b = solver.bisection(mid, S, K, T, r1, "call")
                iv_n, it_n = solver.newton(mid, S, K, T, r1, "call")
            except Exception:
                continue

            rows.append({
                "Symbol": symbol,
                "AsOf": asof,
                "Spot": S,
                "Expiry": exp,
                "T_years": T,
                "T_days": T * 365.0,
                "Strike": K,
                "Moneyness": m,
                "Class": cls,
                "Bid": float(opt["bid"]),
                "Ask": float(opt["ask"]),
                "Mid": mid,
                "IV_Bisection": iv_b,
                "IV_Newton": iv_n,
                "Iter_Bisection": it_b,
                "Iter_Newton": it_n,
                "Volume": float(opt.get("volume", np.nan)) if "volume" in opt else np.nan,
                "OpenInterest": float(opt.get("openInterest", np.nan)) if "openInterest" in opt else np.nan,
            })

    iv_df = pd.DataFrame(rows)
    if iv_df.empty:
        print("[warn] nothing made it into the IV table (filters too strict?)")
        return iv_df

    iv_df.to_csv(os.path.join(exports_dir, f"iv_table_{symbol}_DATA1.csv"), index=False)

    # ATM per expiry is strike closest to S
    atm_rows = []
    for exp, grp in iv_df.groupby("Expiry"):
        g = grp.copy()
        g["abs_diff"] = (g["Strike"] - S).abs()
        atm_rows.append(g.sort_values("abs_diff").iloc[0])
    atm_df = pd.DataFrame(atm_rows).drop(columns=["abs_diff"], errors="ignore")
    atm_df.to_csv(os.path.join(exports_dir, f"atm_iv_{symbol}_DATA1.csv"), index=False)

    # avg by expiry + bucket
    avg_df = iv_df.groupby(["Expiry", "Class"], as_index=False).agg(
        Avg_IV=("IV_Bisection", "mean"),
        N=("IV_Bisection", "size")
    )
    avg_df.to_csv(os.path.join(exports_dir, f"avg_iv_by_bucket_{symbol}_DATA1.csv"), index=False)

    print("\nATM (closest strike) by expiry:")
    print(atm_df[["Expiry", "Strike", "IV_Bisection", "IV_Newton"]].to_string(index=False))

    print("\nAvg IV by expiry + bucket (bisection):")
    print(avg_df.to_string(index=False))

    return iv_df


def put_call_parity_prob9(data1, symbol, r1, exports_dir):
    print("\n" + "=" * 80)
    print(f"[PART 2] PUT-CALL PARITY CHECK (DATA1) — {symbol}")
    print("=" * 80)

    if symbol not in data1 or "calls" not in data1[symbol] or "puts" not in data1[symbol]:
        print("[warn] missing calls/puts")
        return None

    S = data1[symbol]["price"]
    asof = data1[symbol]["date"]

    rows = []
    for calls_df, puts_df, exp in zip(data1[symbol]["calls"], data1[symbol]["puts"], data1[symbol]["expiries"]):
        T = time_to_maturity(exp, asof)

        calls = calls_df.copy()
        puts = puts_df.copy()

        # mid prices
        calls["mid_call"] = (calls["bid"].fillna(0) + calls["ask"].fillna(0)) / 2.0
        puts["mid_put"] = (puts["bid"].fillna(0) + puts["ask"].fillna(0)) / 2.0

        puts_by_strike = puts.set_index("strike")

        for _, c in calls.iterrows():
            K = float(c["strike"])
            call_mid = float(c["mid_call"])
            if call_mid <= 0:
                continue

            implied_put = parity_put_from_call(call_mid, S, K, T, r1)

            put_bid = np.nan
            put_ask = np.nan
            put_mid = np.nan
            if K in puts_by_strike.index:
                rowp = puts_by_strike.loc[K]
                if isinstance(rowp, pd.DataFrame):
                    rowp = rowp.iloc[0]
                put_bid = float(rowp.get("bid", np.nan))
                put_ask = float(rowp.get("ask", np.nan))
                put_mid = float(rowp.get("mid_put", np.nan))

            rows.append({
                "Symbol": symbol,
                "AsOf": asof,
                "Expiry": exp,
                "T_years": T,
                "Strike": K,
                "Spot": S,
                "Call_mid": call_mid,
                "Put_implied": implied_put,
                "Put_bid": put_bid,
                "Put_ask": put_ask,
                "Put_mid": put_mid,
                "Implied_minus_mid": implied_put - put_mid if not np.isnan(put_mid) else np.nan
            })

    df = pd.DataFrame(rows)
    df.to_csv(os.path.join(exports_dir, f"parity_check_{symbol}_DATA1.csv"), index=False)
```

```python

    print("Saved parity CSV. First few rows:")
    print(df.head(10).to_string(index=False))
    return df


def vol_smile_plot_prob10(iv_df, symbol, exports_dir):
    if iv_df is None or iv_df.empty:
        return

    print("\n" + "=" * 80)
    print(f"[PART 2] VOL SMILE PLOT — {symbol}")
    print("=" * 80)

    expiries = sorted(iv_df["Expiry"].unique())[:3]
    colors = ["tab:blue", "tab:red", "tab:green"]

    plt.figure(figsize=(12, 7))
    for i, exp in enumerate(expiries):
        sub = iv_df[iv_df["Expiry"] == exp].sort_values("Strike")
        label = f"{exp} ({sub['T_days'].iloc[0]:.0f}d)"
        plt.scatter(sub["Strike"], sub["IV_Bisection"], s=70, alpha=0.65, color=colors[i], label=label)
        plt.plot(sub["Strike"], sub["IV_Bisection"], linestyle="--", alpha=0.25, color=colors[i])

    plt.title(f"Volatility Smile — {symbol}", fontweight="bold")
    plt.xlabel("Strike (K)")
    plt.ylabel("Implied Volatility (o)")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()

    fn = os.path.join(exports_dir, f"volatility_smile_{symbol}.png")
    plt.savefig(fn, dpi=300, bbox_inches="tight")
    plt.close()
    print("✓ saved", fn)


def bonus_surface_plot_3d(iv_df, symbol, exports_dir):
    if iv_df is None or iv_df.empty:
        return

    print("\n" + "=" * 80)
    print(f"[BONUS] 3D VOL SURFACE — {symbol}")
    print("=" * 80)

    from mpl_toolkits.mplot3d import Axes3D

    fig = plt.figure(figsize=(14, 10))
    ax = fig.add_subplot(111, projection="3d")

    strikes = iv_df["Strike"].values
    days = iv_df["T_days"].values
    ivs = iv_df["IV_Bisection"].values
                                                    #axis lable
    sc = ax.scatter(strikes, days, ivs, c=ivs, cmap="viridis", s=45, alpha=0.7)
    ax.set_xlabel("Strike")
    ax.set_ylabel("Days to Expiry")
    ax.set_zlabel("IV")
    ax.set_title(f"Vol Surface — {symbol}", fontweight="bold")

    cbar = fig.colorbar(sc, ax=ax, shrink=0.6, aspect=7)
    cbar.set_label("IV")

    ax.view_init(elev=20, azim=45)
    plt.tight_layout()

    fn = os.path.join(exports_dir, f"volatility_surface_3d_{symbol}.png")
    plt.savefig(fn, dpi=300, bbox_inches="tight")
    plt.close()
    print("✓ saved", fn)


def greeks_table_prob11(exports_dir):
    print("\n" + "=" * 80)
    print("[PART 2] GREEKS TABLE (Analytical vs Numerical)")
    print("=" * 80)

    # fixed parameters for reproduction
    S, K, T, r, sigma = 100.0, 105.0, 0.5, 0.05, 0.25

    rows = [
        {"Greek": "Delta", "Analytical": Greeks.delta(S, K, T, r, sigma, "call"), "Numerical": Greeks.delta_cd(S, K, T, r, sigma, "
        {"Greek": "Gamma", "Analytical": Greeks.gamma(S, K, T, r, sigma), "Numerical": Greeks.gamma_cd(S, K, T, r, sigma, "call")},
        {"Greek": "Vega", "Analytical": Greeks.vega(S, K, T, r, sigma), "Numerical": Greeks.vega_cd(S, K, T, r, sigma, "call")},
    ]
    df = pd.DataFrame(rows)
    df["Abs_Diff"] = (df["Analytical"] - df["Numerical"]).abs()

    print(df.to_string(index=False))
    df.to_csv(os.path.join(exports_dir, "greeks_table.csv"), index=False)


def day2_pred_prob12(data2, iv_df_day1, symbol, r2, exports_dir):
    print("\n" + "=" * 80)
    print(f"[PART 2] PROBLEM 12 — DAY 2 PRICING USING DAY 1 IV — {symbol}")
    print("=" * 80)

    if iv_df_day1 is None or iv_df_day1.empty:
        print("[warn] no Day1 IV table => can't do Day2 pricing")
        return None
    if symbol not in data2 or "calls" not in data2[symbol]:
        print("[warn] no Day2 calls")
        return None

    S2 = data2[symbol]["price"]
    asof2 = data2[symbol]["date"]
    print(f"Day2 asof={asof2}  S2=${S2:.2f}  r2={r2:.4%}")

    #lookup Day 1 IV (Expiry, Strike)
    iv_lookup = iv_df_day1.set_index(["Expiry", "Strike"])["IV_Bisection"].to_dict()

    rows = []
    for calls_df, exp in zip(data2[symbol]["calls"], data2[symbol]["expiries"]):
        T = time_to_maturity(exp, asof2)

        df = calls_df.copy()
        df = df[(df["bid"] > 0) & (df["ask"] > 0)].copy()  #for day 2  just need a mid
        for _, opt in df.iterrows():
            K = float(opt["strike"])
            mid2 = float((opt["bid"] + opt["ask"]) / 2.0)

            iv1 = iv_lookup.get((exp, K))
            if iv1 is None:
                continue

            pred = BS.call(S2, K, T, r2, float(iv1))
            err = mid2 - pred
            pct = (err / mid2 * 100.0) if mid2 > 0 else np.nan

            rows.append({
                "Symbol": symbol,
                "AsOf_Day2": asof2,
                "Expiry": exp,
                "T_years": T,
                "Strike": K,
                "Spot_Day2": S2,
                "IV_from_Day1": float(iv1),
                "Actual_mid_Day2": mid2,
                "Predicted_BS": pred,
                "Error": err,
                "Pct_Error": pct,
            })

    pred_df = pd.DataFrame(rows)
    if pred_df.empty:
        print("[warn] no matching (expiry,strike) between Day1 IVs and Day2 chain")
        return None

    pred_df.to_csv(os.path.join(exports_dir, f"day2_predictions_{symbol}.csv"), index=False)
                    #cmd line argument setup
    print("Accuracy summary:")
    print(f"  MAE  = ${pred_df['Error'].abs().mean():.4f}")
    print(f"  MAPE = {pred_df['Pct_Error'].abs().mean():.2f}%")
    print(f"  RMSE = ${np.sqrt((pred_df['Error']**2).mean()):.4f}")

    # plot
    plt.figure(figsize=(10, 6))
    plt.scatter(pred_df["Actual_mid_Day2"], pred_df["Predicted_BS"], alpha=0.6)
    mn = min(pred_df["Actual_mid_Day2"].min(), pred_df["Predicted_BS"].min())
    mx = max(pred_df["Actual_mid_Day2"].max(), pred_df["Predicted_BS"].max())
    plt.plot([mn, mx], [mn, mx], "r--", label="perfect")
    plt.title(f"Day 2 Price Prediction — {symbol}")
    plt.xlabel("Actual Day 2 mid")
    plt.ylabel("Predicted (Day 1 IV)")
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()
    fn = os.path.join(exports_dir, f"day2_prediction_{symbol}.png")
    plt.savefig(fn, dpi=300, bbox_inches="tight")
    plt.close()
    print("✓ saved", fn)

    return pred_df




# ================================
# MAIN
# ================================
def parse_args():
    p = argparse.ArgumentParser()
    p.add_argument("--symbols", nargs="*", default=["TSLA", "SPY", "^VIX"])
    p.add_argument("--expiries", type=int, default=3)
    p.add_argument("--r1", type=float, default=0.0433, help="rate for DATA1")
    p.add_argument("--r2", type=float, default=0.0433, help="rate for DATA2")
    p.add_argument("--atm-lo", type=float, default=0.95)
    p.add_argument("--atm-hi", type=float, default=1.05)
    p.add_argument("--exports", type=str, default="exports")
    p.add_argument("--no-download", action="store_true")
    p.add_argument("--no-plots", action="store_true")
    p.add_argument("--no-3d", action="store_true")
    p.add_argument("--quiet", action="store_true")
    return p.parse_args()


def main():
    args = parse_args()
    os.makedirs(args.exports, exist_ok=True)
```

```python
    print("=" * 80)
    print("FE621 HOMEWORK 1 - COMPUTATIONAL FINANCE")
    print("Student: Julius Stiemer")
    print("=" * 80)
    print("Run date:", datetime.now().strftime("%Y-%m-%d"))
    print("Exports folder:", os.path.abspath(args.exports))

    data1, data2 = {}, {}

    if not args.no_download:
        data1, data2 = get_data1_data2(
            symbols=args.symbols,
            expiries_to_use=args.expiries,
            exports_dir=args.exports,
            quiet=args.quiet
        )
    else:
        print("\n[no-download] skipping Yahoo download (Part 2 real-data parts won't run).")

    # P2 Greeks table (analytical vs numerical)
    greeks_table_prob11(args.exports)

    # P2 real options tasks for TSLA and SPY
    for sym in [s for s in args.symbols if s != "^VIX"]:
        if sym not in data1:
            continue

        iv_df = iv_table_problems4_8(
            data1, sym, args.r1, args.exports, atm_lo=args.atm_lo, atm_hi=args.atm_hi
        )

        put_call_parity_prob9(data1, sym, args.r1, args.exports)

        if (not args.no_plots) and iv_df is not None and (not iv_df.empty):
            vol_smile_plot_prob10(iv_df, sym, args.exports)
            if not args.no_3d:
                bonus_surface_plot_3d(iv_df, sym, args.exports)

        if sym in data2 and iv_df is not None and (not iv_df.empty):
            day2_pred_prob12(data2, iv_df, sym, args.r2, args.exports)



if __name__ == "__main__":
    main()
```