SPX Premarket Dashboard - Instructions & Script

This PDF contains: 1) Quick prerequisites and install steps. 2) How to run the Streamlit dashboard. 3) Full Python script (copy & paste into a .py file if needed).

Prerequisites: - Python 3.8+ recommended. - Install required packages (example): pip install streamlit yfinance pandas numpy plotly pandas_ta requests reportlab

Run steps: 1) Save the script file as spx_premarket_dashboard.py (already included). 2) From a terminal: streamlit run spx_premarket_dashboard.py 3) The dashboard will open in a browser (http://localhost:8501 by default).

Notes on options data: - This script can display options data only if you provide an options API. - OptionCharts.io and MarketChameleon provide visual OI heatmaps — the script can be used alongside them.

Full script (copy/paste):

```
# Filename: spx_premarket_dashboard.py
# Run with: streamlit run spx_premarket_dashboard.py
# Description: Streamlit dashboard to show SPX/ES premarket indicators, support/resistance, VWAP, pivots, volume
# and optional options OI/IV integration (API required for options).
#
# Dependencies:
# pip install streamlit yfinance pandas numpy plotly pandas_ta requests

import streamlit as st
import yfinance as yf
import pandas as pd
import numpy as np
import datetime as dt
import plotly.graph_objects as go
import requests

# Try pandas_ta for ATR; fallback to manual ATR if unavailable
try:
    import pandas_ta as ta
    PANDAS_TA_AVAILABLE = True
except Exception:
    PANDAS_TA_AVAILABLE = False

# ---------------------------
# Config (edit as desired)
# ---------------------------
st.set_page_config(layout="wide", page_title="SPX Premarket Dashboard")
TICK_SPX = "^GSPC"    # SPX proxy; you can also use "SPY"
TICK_FUT = "ES=F"     # E-mini S&P futures (Yahoo)
TICK_VIX = "^VIX"
DEFAULT_LOOKBACK_DAYS = 10

# Optional: options API config (leave empty to skip)
OPTIONS_API_ENABLED = False
OPTIONS_API_URL = "https://YOUR_OPTION_API_ENDPOINT"
OPTIONS_API_KEY = "YOUR_API_KEY"

# ---------------------------
# Helper functions
# ---------------------------
@st.cache_data(ttl=60)
def fetch_yahoo_history(ticker, period="5d", interval="5m"):
    """Fetch history with yfinance. Returns DataFrame with 'Datetime' column."""
    try:
        tk = yf.Ticker(ticker)
        df = tk.history(period=period, interval=interval, actions=False, auto_adjust=False)
        if df.empty:
            # fallback
            df = tk.history(period="30d", interval="15m", actions=False, auto_adjust=False)
        df = df.reset_index().rename(columns={"index":"Datetime"})
        return df
    except Exception as e:
        st.error(f"Error fetching {ticker}: {e}")
        return pd.DataFrame()

@st.cache_data(ttl=60)
```

```
def fetch_snapshot(ticker):
    """Fetch a lightweight snapshot (last close)"""
    try:
        tk = yf.Ticker(ticker)
        price = tk.history(period="1d", interval="1m")
        last = None
        if not price.empty:
            last = price['Close'].iloc[-1]
        return {"last": last}
    except Exception:
        return {"last": None}

def compute_vwap(df, price_col="Close", vol_col="Volume"):
    """Cumulative VWAP for the df (assumes chronological order)."""
    if df.empty or price_col not in df.columns or vol_col not in df.columns:
        return pd.Series(dtype=float)
    p = df[price_col].fillna(method='ffill')
    v = df[vol_col].fillna(0)
    cum_pv = (p * v).cumsum()
    cum_v = v.cumsum().replace(0, np.nan)
    vwap = (cum_pv / cum_v).fillna(method='ffill')
    return vwap

def pivot_points_classic(prev_row):
    """Classic pivot points from a single previous-day row (expects High, Low, Close)"""
    H = prev_row['High']
    L = prev_row['Low']
    C = prev_row['Close']
    P = (H + L + C) / 3.0
    R1 = 2*P - L
    S1 = 2*P - H
    R2 = P + (H - L)
    S2 = P - (H - L)
    R3 = H + 2*(P - L)
    S3 = L - 2*(H - P)
    return dict(P=P, R1=R1, S1=S1, R2=R2, S2=S2, R3=R3, S3=S3)

def support_resistance_from_profile(df, n_bins=40):
    """Simple volume profile: find price bins with highest traded volume."""
    if df.empty or 'Close' not in df.columns or 'Volume' not in df.columns:
        return []
    prices = df['Close'].values
    vols = df['Volume'].values
    if len(prices) < 2:
        return []
    bins = np.linspace(prices.min(), prices.max(), n_bins)
    bin_idx = np.digitize(prices, bins)
    vol_by_bin = pd.Series(0, index=range(1, len(bins)+1), dtype=float)
    for i, vol in zip(bin_idx, vols):
        if 1 <= i <= len(bins):
            vol_by_bin[i] += vol
    top_bins = vol_by_bin.sort_values(ascending=False).head(6).index
    levels = [ (bins[i-1] + (bins[i] if i < len(bins) else bins[-1]))/2 for i in top_bins ]
    return sorted(levels)

def compute_atr(df, length=14):
    """Simple ATR calculation (fallback if pandas_ta not available)."""
    if df.empty or not {'High','Low','Close'}.issubset(df.columns):
        return np.nan
    high = df['High']
    low = df['Low']
    close = df['Close']
    tr1 = high - low
    tr2 = (high - close.shift(1)).abs()
    tr3 = (low - close.shift(1)).abs()
    tr = pd.concat([tr1, tr2, tr3], axis=1).max(axis=1)
    atr = tr.rolling(window=length).mean()
    return atr.iloc[-1] if not atr.empty else np.nan

# Optional: user-supplied options API integration (must return DataFrame with strike, call_oi, put_oi, call_iv,
def fetch_options_data_api(expiry=None):
    if not OPTIONS_API_ENABLED:
        return None
    headers = {"Authorization": f"Bearer {OPTIONS_API_KEY}"}
    params = {}
    if expiry:
        params["expiry"] = expiry
    resp = requests.get(OPTIONS_API_URL, headers=headers, params=params, timeout=10)
    if resp.status_code == 200:
        d = resp.json()
```

```
        df = pd.DataFrame(d)
        return df
    else:
        st.warning(f"Options API error: {resp.status_code} {resp.text}")
        return None

# ---------------------------
# UI: Sidebar controls
# ---------------------------
st.sidebar.header("Settings")
ticker_spx = st.sidebar.text_input("SPX proxy ticker", TICK_SPX)
ticker_fut = st.sidebar.text_input("Futures ticker (ES)", TICK_FUT)
ticker_vix = st.sidebar.text_input("VIX ticker", TICK_VIX)
lookback_days = st.sidebar.number_input("History lookback (days)", value=DEFAULT_LOOKBACK_DAYS, min_value=2, ma
use_5m = st.sidebar.checkbox("Use 5m resolution for history (faster)", value=True)
st.sidebar.markdown("---")
st.sidebar.write("Options OI/IV (optional)")
opt_enabled = st.sidebar.checkbox("Enable options API integration (requires editing script)", value=False)

# ---------------------------
# Main fetch & compute
# ---------------------------
res_interval = "5m" if use_5m else "1m"
period = f"{max(lookback_days, 5)}d"

with st.spinner("Fetching market data..."):
    df_fut = fetch_yahoo_history(ticker_fut, period=period, interval=res_interval)
    df_spx = fetch_yahoo_history(ticker_spx, period=period, interval=res_interval)
    df_vix = fetch_yahoo_history(ticker_vix, period=period, interval=res_interval)
    snap_fut = fetch_snapshot(ticker_fut)
    snap_spx = fetch_snapshot(ticker_spx)
    snap_vix = fetch_snapshot(ticker_vix)

# Quick metrics
col1, col2, col3 = st.columns(3)
if snap_fut["last"] is not None:
    col1.metric(f"{ticker_fut} (Futures)", f"{snap_fut['last']:.2f}")
else:
    col1.write(f"{ticker_fut}: no snapshot")

if snap_spx["last"] is not None:
    col2.metric(f"{ticker_spx}", f"{snap_spx['last']:.2f}")
else:
    col2.write(f"{ticker_spx}: no snapshot")

if snap_vix["last"] is not None:
    col3.metric("VIX", f"{snap_vix['last']:.2f}")
else:
    col3.write("VIX: no snapshot")

# ---------------------------
# Compute session levels and indicators
# ---------------------------
def compute_session_levels(df):
    if df.empty:
        return {}
    df = df.copy()
    if 'Datetime' in df.columns:
        df['dt'] = pd.to_datetime(df['Datetime'])
    else:
        df['dt'] = pd.to_datetime(df.iloc[:,0])
    df = df.sort_values('dt')
    df['date_only'] = df['dt'].dt.date
    unique_dates = df['date_only'].unique()
    if len(unique_dates) >= 2:
        prev_date = unique_dates[-2]
        today_date = unique_dates[-1]
    else:
        prev_date = unique_dates[-1]
        today_date = unique_dates[-1]
    prev_day_df = df[df['date_only'] == prev_date]
    today_df = df[df['date_only'] == today_date]
    prev_summary = {}
    if not prev_day_df.empty:
        prev_summary['prev_high'] = prev_day_df['High'].max() if 'High' in prev_day_df else prev_day_df['Close']
        prev_summary['prev_low']  = prev_day_df['Low'].min() if 'Low' in prev_day_df else prev_day_df['Close'].m
        prev_summary['prev_close']= prev_day_df['Close'].iloc[-1]
        prev_summary['prev_row'] = prev_day_df.iloc[-1]
    else:
        prev_summary['prev_high'] = None
```

```
    overnight_high = df[df['date_only'] == prev_date]['Close'].max() if not df[df['date_only'] == prev_date].em
    overnight_low = df[df['date_only'] == prev_date]['Close'].min() if not df[df['date_only'] == prev_date].emp
    return {"prev": prev_summary, "overnight_high": overnight_high, "overnight_low": overnight_low, "prev_date"

levels_fut = compute_session_levels(df_fut)
levels_spx = compute_session_levels(df_spx)

# VWAP
if not df_fut.empty:
    df_fut['vwap'] = compute_vwap(df_fut, price_col='Close', vol_col='Volume')
else:
    df_fut['vwap'] = pd.Series(dtype=float)

# Technicals
def compute_technicals(df):
    out = {}
    if df.empty:
        return out
    series = df['Close']
    out['ma50'] = series.rolling(window=50).mean().iloc[-1] if len(series) >= 50 else np.nan
    out['ma200'] = series.rolling(window=200).mean().iloc[-1] if len(series) >= 200 else np.nan
    if PANDAS_TA_AVAILABLE:
        out['atr14'] = ta.atr(df['High'], df['Low'], df['Close'], length=14).iloc[-1]
    else:
        out['atr14'] = compute_atr(df, length=14)
    return out

tech_fut = compute_technicals(df_fut)
tech_spx = compute_technicals(df_spx)

# Volume profile support/resistance
vp_levels = support_resistance_from_profile(df_fut)

# Pivot points
pivots = {}
if levels_fut.get('prev', {}).get('prev_row') is not None:
    pivots = pivot_points_classic(levels_fut['prev']['prev_row'])
else:
    pivots = {}

# ---------------------------
# Sentiment rule engine
# ---------------------------
sentiments = []
if levels_fut.get('prev', {}).get('prev_close') is not None and snap_fut["last"] is not None:
    prev_close = levels_fut['prev']['prev_close']
    gap = (snap_fut['last'] - prev_close) / prev_close * 100
    if gap > 0.2:
        sentiments.append(("GAP", "Bullish", f"Futures gap +{gap:.2f}%"))
    elif gap < -0.2:
        sentiments.append(("GAP", "Bearish", f"Futures gap {gap:.2f}%"))
    else:
        sentiments.append(("GAP", "Neutral", f"Futures gap {gap:.2f}%"))

if not df_fut.empty and 'vwap' in df_fut.columns and not df_fut['vwap'].isna().all():
    last_price = df_fut['Close'].iloc[-1]
    last_vwap = df_fut['vwap'].iloc[-1]
    if last_price > last_vwap:
        sentiments.append(("VWAP", "Bullish", f"Price above VWAP ({last_price:.2f} > {last_vwap:.2f})"))
    else:
        sentiments.append(("VWAP", "Bearish", f"Price below VWAP ({last_price:.2f} < {last_vwap:.2f})"))

if not np.isnan(tech_fut.get('ma50', np.nan)) and not np.isnan(tech_fut.get('ma200', np.nan)):
    if tech_fut['ma50'] > tech_fut['ma200']:
        sentiments.append(("MA", "Bullish", "50MA > 200MA"))
    else:
        sentiments.append(("MA", "Bearish", "50MA < 200MA"))

atr = tech_fut.get('atr14', np.nan)
if not np.isnan(atr):
    sentiments.append(("ATR", "Info", f"ATR(14): {atr:.2f}"))

# ---------------------------
# Display dashboard panes
# ---------------------------
st.markdown("## SPX Premarket Dashboard (Price, Futures & Options Overlay)")

left_col, right_col = st.columns([3,2])

with left_col:
```

```
    st.subheader(f"{ticker_fut} (Futures) - Live snapshot")
    if df_fut.empty:
        st.write("No futures data available.")
    else:
        fig = go.Figure()
        fig.add_trace(go.Candlestick(x=df_fut['Datetime'], open=df_fut['Open'], high=df_fut['High'],
                                     low=df_fut['Low'], close=df_fut['Close'], name="Price"))
        # VWAP
        if 'vwap' in df_fut.columns:
            fig.add_trace(go.Scatter(x=df_fut['Datetime'], y=df_fut['vwap'], name="VWAP", line=dict(width=1)))
        # Plot pivot levels
        for k,v in pivots.items():
            fig.add_hline(y=v, line_dash="dash", annotation_text=k, annotation_position="right")
        # Volume profile levels
        for lvl in vp_levels:
            fig.add_hline(y=lvl, line_dash="dot", annotation_text=f"VP {lvl:.2f}", annotation_position="left")
        fig.update_layout(height=520, margin=dict(l=10,r=10,t=30,b=10), xaxis_rangeslider_visible=False)
        st.plotly_chart(fig, use_container_width=True)

with right_col:
    st.subheader("Quick Market Signals")
    for item in sentiments:
        tag, bias, text = item
        color = "green" if bias == "Bullish" else ("red" if bias == "Bearish" else "orange")
        st.markdown(f"**{tag}**: <span style='color:{color}'>{bias}</span> — {text}", unsafe_allow_html=True)

    st.markdown("---")
    st.write("Key levels (from previous session & VP):")
    if levels_fut.get('prev', {}).get('prev_high') is not None:
        st.write(f"- Prev High {levels_fut['prev']['prev_high']:.2f}")
        st.write(f"- Prev Low {levels_fut['prev']['prev_low']:.2f}")
        st.write(f"- Prev Close {levels_fut['prev']['prev_close']:.2f}")
    if levels_fut.get('overnight_high'):
        st.write(f"- Overnight High {levels_fut['overnight_high']:.2f}")
        st.write(f"- Overnight Low {levels_fut['overnight_low']:.2f}")

    if vp_levels:
        st.write("- Volume Profile top levels: " + ", ".join([f"{v:.2f}" for v in vp_levels]))

# Below: SPX/ETF chart and VIX
sp_col, vix_col = st.columns([3,1])

with sp_col:
    st.subheader(f"{ticker_spx} recent")
    if df_spx.empty:
        st.write("No SPX history available.")
    else:
        fig2 = go.Figure()
        fig2.add_trace(go.Candlestick(x=df_spx['Datetime'], open=df_spx['Open'], high=df_spx['High'],
                                      low=df_spx['Low'], close=df_spx['Close'], name=ticker_spx))
        # simple 50/200MA
        if 'Close' in df_spx.columns:
            df_spx['ma50'] = df_spx['Close'].rolling(50).mean()
            df_spx['ma200'] = df_spx['Close'].rolling(200).mean()
            fig2.add_trace(go.Scatter(x=df_spx['Datetime'], y=df_spx['ma50'], name="MA50", line=dict(width=1)))
            fig2.add_trace(go.Scatter(x=df_spx['Datetime'], y=df_spx['ma200'], name="MA200", line=dict(width=1)
        fig2.update_layout(height=380, margin=dict(l=10,r=10,t=30,b=10), xaxis_rangeslider_visible=False)
        st.plotly_chart(fig2, use_container_width=True)

with vix_col:
    st.subheader("VIX")
    if df_vix.empty:
        st.write("No VIX data.")
    else:
        vfig = go.Figure()
        vfig.add_trace(go.Scatter(x=df_vix['Datetime'], y=df_vix['Close'], name="VIX"))
        vfig.update_layout(height=380, margin=dict(l=10,r=10,t=30,b=10))
        st.plotly_chart(vfig, use_container_width=True)

# Options panel (optional)
st.markdown("---")
st.subheader("Options Overview (optional)")
if opt_enabled and OPTIONS_API_ENABLED:
    st.write("Fetching options via configured API...")
    opt_df = fetch_options_data_api()
    if opt_df is None or opt_df.empty:
        st.write("No options data returned from API.")
    else:
        st.write("Top strikes by open interest (sample):")
        st.dataframe(opt_df.sort_values(by=['call_oi','put_oi'], ascending=False).head(20))
```

```
else:
    st.write("Options integration is disabled. To enable, set 'OPTIONS_API_ENABLED = True' in the script and pr
    st.write("You can still use external tools (OptionCharts.io, MarketChameleon) for detailed options OI heatma

# Footer notes & quick export
st.markdown("---")
st.write("Notes:")
st.write("- This dashboard is a starting point. For production usage, wire a paid options API and improve error
st.write("- If Yahoo/yiannce limits 1m data, use 5m for stable results.")
st.write("- To run: install dependencies, then `streamlit run spx_premarket_dashboard.py`")
```