US TREASURY YEILD CURVE PARSER that retrieves data directly from the Federal Reserve Economic Data (FRED) database. (https://fred.stlouisfed.org/)

Python based Parser functionality:

Yield Data Retrieval, Forward Curve calculation, Spreads, Correlations and Recession probability indicator

Short overview:

U.S. Treasury yield curve reflects market expectations for interest rates, inflation and growth. The **2s10s spread** is a widely used recession indicator, often signalling downturns 12–18 months ahead, while the **5s30s** spread reflects long-term economic expectations.

Forward rates are exceptionally important for understanding the structure of the yield curve and for identifying relatively cheap or rich segments along the curve. The forward curve serves as a benchmark for market expectations, reflecting both hopes and fears. Implied **one-year forward spot rates** represent the future spot rates that equalize holding-period returns across government bonds over the coming years. Forward rates indicate how much yields on longer-term bonds must change to offset returns available on shorter-term maturities. Forwards tell how much yield along term bond need to change to offset yield over short term.

Generally, **positive carry** (causing against rising rates) / **negative carry** (future flattening of spot curve) offset negative carry.

(HINT) **The Forward curve is benchmark for hopes & fears.**

Assuming **annual compounding**, the one-year forward rate is easily computed:

$$(1 + f_{m,n})^{n-m} = \frac{(1 + s_n)^n}{(1 + s_m)^m}$$

One-year forward rates measure reward for lengthening the maturity of investment by one year, while spot rates measure an investment average reward from today to maturity n.

Parser Solution Idea & Technik:

Recently I found out very interesting Python library : **pandas_datareader.data** is a module from the pandas-data reader library that lets you download financial and macroeconomic data directly from online data sources into pandas DataFrames. It allows to download Treasury yields directly from Federal Reserve Economic Data (FRED).

It enables powerful and absolutely free parser for **Fixed income portfolio managers and traders.** The parser built on **Streamlit** library, which allows user to plot curve and statistics directly to Web browser and make necessary dialogs in windows designed. **Streamlit** is a Python framework for quickly building interactive web applications, especially for data analysis and visualization.

Following imports are necessary to run the code:

import streamlit as st

import pandas as pd

import pandas_datareader.data as web

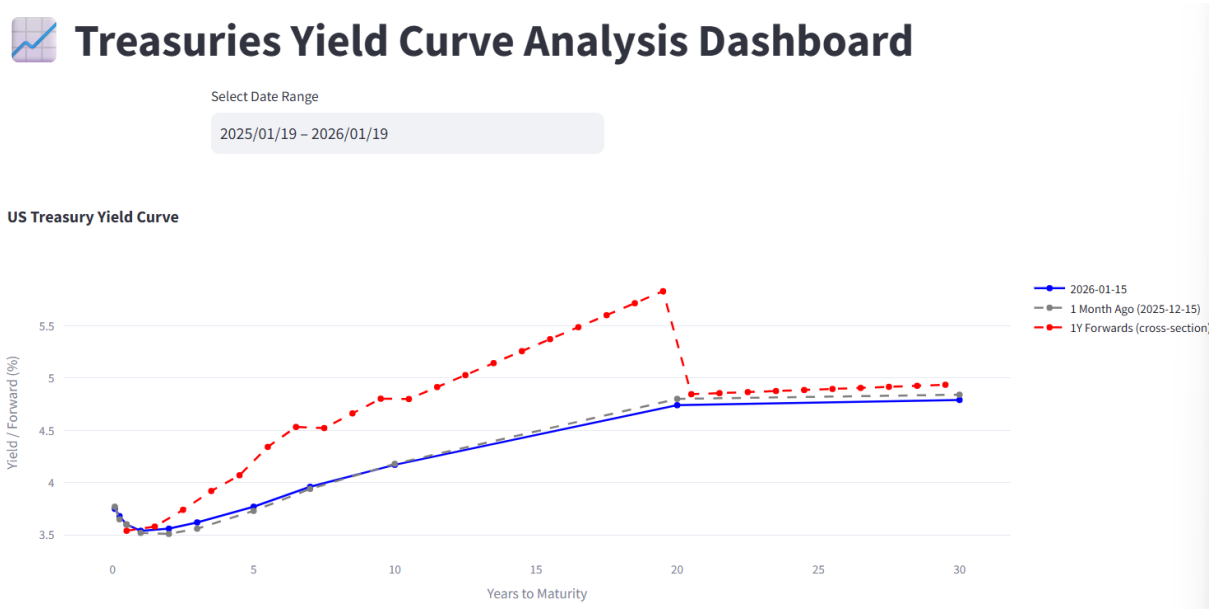
import numpy as np

import plotly.graph_objects as go

from datetime import datetime, timedelta

***Results and plots*:**

Firstly, the Parser plots '**Treasuries Yield Curve Analysis Dashboard**' which representation of **3 main curves**: Current **Spot** Yield + Yield 1 **Month ago** (adjustable by the user) & (**1Y Implied One-Year Forward curve**)



The forward curve typically lies above the spot curve when the yield curve is upward sloping. When the yield curve is inverted, this relationship reverses and the forward curve lies below the spot curve

One-year forward rates measure the marginal reward lengthening the maturity of investments by one year. It is useful to view rates as **break-even rates**.

Generally, if the yield changes implied by the forward rates are realized, all government bonds, regardless of maturity, earn the same holding-period return. Historically the bond risk premia is not linear in duration but increase steeply with duration in the front end of the curve and much more slowly after two years.

We often hear that ‘forward rates shows the market expectation of future rates’. However, this statement is only true if no bon risk premia exist and convexity is too small. **The amount of spread change implied by the forwards is a useful indicator or yield curve trades.** The forwards tell how much the yield of given bond (a longer term bond) need to change to offset an initial yield over the short-term rate.

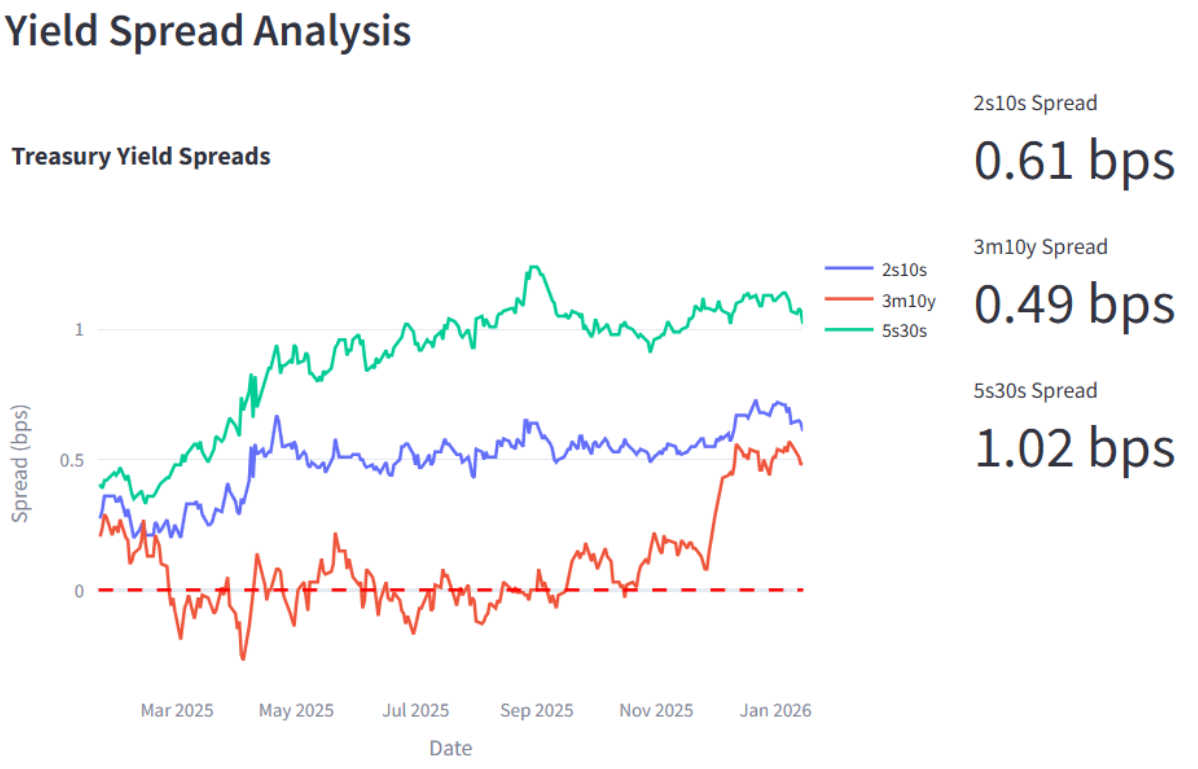
Pls check forward curve (red) lying above spot with cheap sectors in belly and long -end of the curve.

This parser generates the 1-year forward curve in a spreadsheet.

Show 1-Year Forward Curve Data

DATE	0Y-1Y	1Y-2Y	2Y-3Y	3Y-4Y	4Y-5Y	5Y-6Y	6Y-7Y	7Y-8Y	8Y-9Y	9Y-10Y	10Y-11Y	11Y-12Y	12Y-13Y	13Y-14Y	14Y-15Y	15Y-16Y	16Y-17Y
2026-01-02 00:00:00	3.47	3.47	3.7102	3.9305	4.1209	4.3716	4.5822	4.5917	4.7522	4.9128	4.874	4.9984	5.1229	5.2474	5.3719	5.4964	5.621
2026-01-05 00:00:00	3.47	3.45	3.6701	3.8905	4.0708	4.3416	4.5522	4.5885	4.7557	4.923	4.854	4.9784	5.1029	5.2274	5.3519	5.4764	5.601
2026-01-06 00:00:00	3.48	3.46	3.6801	3.9005	4.0808	4.3516	4.5622	4.5985	4.7657	4.933	4.864	4.9884	5.1129	5.2374	5.3619	5.4864	5.611
2026-01-07 00:00:00	3.48	3.46	3.6501	3.8704	4.0407	4.3316	4.5422	4.5517	4.7122	4.8728	4.823	4.9454	5.0678	5.1903	5.3128	5.4353	5.5579
2026-01-08 00:00:00	3.48	3.5	3.7001	3.9205	4.1008	4.3716	4.5822	4.5917	4.7522	4.9128	4.8519	4.9723	5.0927	5.2131	5.3336	5.4541	5.5747
2026-01-09 00:00:00	3.52	3.56	3.6901	3.9104	4.0706	4.3514	4.552	4.5649	4.7187	4.8725	4.8198	4.9361	5.0525	5.1689	5.2854	5.4019	5.5184
2026-01-12 00:00:00	3.53	3.55	3.6901	3.9505	4.1308	4.3714	4.572	4.5581	4.7052	4.8523	4.8408	4.9592	5.0776	5.196	5.3145	5.433	5.5515
2026-01-13 00:00:00	3.51	3.55	3.65	3.9305	4.1108	4.3514	4.552	4.5649	4.7187	4.8725	4.8308	4.9492	5.0676	5.186	5.3045	5.423	5.5415
2026-01-14 00:00:00	3.5	3.52	3.6601	3.8804	4.0406	4.3214	4.522	4.5349	4.6887	4.8425	4.7898	4.9061	5.0225	5.1389	5.2554	5.3719	5.4884
2026-01-15 00:00:00	3.54	3.58	3.7401	3.9203	4.0705	4.3413	4.5318	4.5213	4.6617	4.8021	4.7987	4.9131	5.0274	5.1418	5.2563	5.3707	5.4852

Yield spread dynamic highlights the main spread dynamics over the selected period, reflecting the recent steepening of the yield curve over the past year.



Forward Rate Analysis

Finally, the tool predicts **Recession Probability Indicator based on 2s10s spread** and makes **Correlation Matrix**

Recession Probability (Based on 2s10s spread)

49.8%

Yield Curve Statistics

Summary Statistics

	1M	3M	6M	1Y	2Y	3Y	5Y	7Y	10Y	20Y
count	260	260	260	260	260	260	260	260	260	260
mean	4.2359	4.1813	4.0666	3.8885	3.7823	3.7813	3.8944	4.0701	4.2757	4.7752
std	0.2178	0.2504	0.2639	0.2393	0.2462	0.2446	0.2284	0.2011	0.1637	0.1296
min	3.65	3.62	3.56	3.47	3.41	3.42	3.55	3.74	3.97	4.44
25%	4.17	4.0075	3.8	3.6475	3.57	3.57	3.71	3.91	4.14	4.68
50%	4.35	4.32	4.21	3.96	3.76	3.735	3.86	4.06	4.265	4.78
75%	4.37	4.35	4.29	4.1	3.96	3.93	4.0325	4.2	4.4	4.87
max	4.49	4.46	4.36	4.3	4.36	4.37	4.48	4.56	4.66	5.08

Correlation Matrix

	1M	3M	6M	1Y	2Y	3Y	5Y	7Y	10Y	20Y
1M	1	0.919	0.8494	0.7887	0.6583	0.5806	0.5444	0.5492	0.555	0.2589
3M	0.919	1	0.9693	0.9079	0.7531	0.6727	0.6464	0.6697	0.6939	0.4179
6M	0.8494	0.9693	1	0.9696	0.8472	0.7833	0.763	0.782	0.7843	0.4468
1Y	0.7887	0.9079	0.9696	1	0.9411	0.8966	0.8799	0.8905	0.8714	0.4761
2Y	0.6583	0.7531	0.8472	0.9411	1	0.9886	0.9774	0.9623	0.8946	0.3989
3Y	0.5806	0.6727	0.7833	0.8966	0.9886	1	0.9929	0.9735	0.8939	0.379
5Y	0.5444	0.6464	0.763	0.8799	0.9774	0.9929	1	0.9889	0.9201	0.4368
7Y	0.5492	0.6697	0.782	0.8905	0.9623	0.9735	0.9889	1	0.9665	0.5598
10Y	0.555	0.6939	0.7843	0.8714	0.8946	0.8939	0.9201	0.9665	1	0.7461
20Y	0.2589	0.4179	0.4468	0.4761	0.3989	0.379	0.4368	0.5598	0.7461	1

Parser Installation:

In order to deploy on local machine.

- Pls make sure **following libraries are installed**:

import streamlit as st

import pandas as pd

import pandas_datareader.data as web

import numpy as np

import plotly.graph_objects as go

from datetime import datetime, timedelta

- Download and Save the code on local PC.

Following link for download full code in Python:

https://github.com/shtyiva/TSU_curve

- Creates a Run_File.bat file that triggers Streamlit to run from the specified code path

(**Example**) streamlit run "C:\Users\Ivan DOC\ ALGO\TSU_curve\TSU_Curve.py

References:

Overview of Forward Rate Analysis (Salomon Brothers)

Streamlit documentation and tutorials (https://docs.streamlit.io/)

Full Code (Python)

```
import streamlit as st
import pandas as pd
import pandas_datareader.data as web
import numpy as np
import plotly.graph_objects as go
from datetime import datetime, timedelta

# Corrected Treasury symbols
TREASURY_SYMBOLS = {
    "1M": "DGS1MO",  # 1 Month
    "3M": "DGS3MO",  # 3 Months
    "6M": "DGS6MO",  # 6 Months
    "1Y": "DGS1",  # 1 Year
    "2Y": "DGS2",  # 2 Years
    "3Y": "DGS3",  # 3 Years
    "5Y": "DGS5",  # 5 Years
    "7Y": "DGS7",  # 7 Years
    "10Y": "DGS10",  # 10 Years
    "20Y": "DGS20",  # 20 Years
    "30Y": "DGS30",  # 30 Years
}



def fetch_yield_data(start_date=None):
    """
    Fetch yield curve data from FRED (Federal Reserve Economic Data).
    Returns a DataFrame with maturities as columns in % yields.
    """
    if start_date is None:
        start_date = datetime.today() - datetime.timedelta(days=365)

    end_date = datetime.today()
    series = []
```

```
    for maturity, symbol in TREASURY_SYMBOLS.items():
        try:
            df = web.DataReader(symbol, "fred", start=start_date, end=end_date)
            # df has one column named by the symbol (e.g., 'DGS10')
            s = df[symbol].rename(maturity)  # rename to '10Y', '2Y', etc.
            series.append(s)
        except Exception as e:
            print(f"Could not fetch data for {maturity} ({symbol}): {e}")
            series.append(pd.Series(name=maturity, dtype="float64"))

    # Align on the date index and combine
    df = pd.concat(series, axis=1)

    # Fill small gaps (FRED has occasional missing days/holidays)
    df = df.ffill().bfill()

    return df


def plot_yield_curve(df, selected_date, maturities, one_year_forward_curve=None):
    """
    Plot the spot yield curve and optionally the 1Y forward curve.
    """
    if df.empty:
        st.error("No data available.")
        return None

    # If no forward curve provided, compute it
    if one_year_forward_curve is None:
        one_year_forward_curve = calculate_one_year_forward_curve(
            df=df,
            maturities=maturities,
            max_year=30,
        )

    # --- your existing code from here on ---
    selected_ts = pd.Timestamp(selected_date)
    available_dates = df.index[df.index <= selected_ts]
    if len(available_dates) == 0:
```

```
    st.error("No data available for the selected date range")
    return None
closest_date = available_dates.max()

# Build spot curve vectors
current = df.loc[closest_date]
xs_spot, ys_spot = [], []
for label, year in maturities.items():
    if (label in df.columns) and pd.notna(current.get(label)):
        xs_spot.append(year)
        ys_spot.append(float(current[label]))

fig = go.Figure()

# Current spot curve
fig.add_trace(go.Scatter(
    x=xs_spot,
    y=ys_spot,
    name=closest_date.strftime('%Y-%m-%d'),
    mode='lines+markers',
    line=dict(color='blue', width=2)
))

# One month ago comparison (unchanged)
month_ago_candidate = closest_date - pd.DateOffset(months=1)
prev_dates = df.index[(df.index <= month_ago_candidate)]
if len(prev_dates) > 0:
    month_ago = prev_dates.max()
    prev = df.loc[month_ago]
    xs_prev, ys_prev = [], []
    for label, year in maturities.items():
        if (label in df.columns) and pd.notna(prev.get(label)):
            xs_prev.append(year)
            ys_prev.append(float(prev[label]))

    if xs_prev:
        fig.add_trace(go.Scatter(
            x=xs_prev,
            y=ys_prev,
```

```
            name=f'1 Month Ago ({month_ago.strftime("%Y-%m-%d")})',
            mode='lines+markers',
            line=dict(color='gray', width=2, dash='dash')
        ))

    # Overlay 1Y forward curve
    if (closest_date in one_year_forward_curve.index):
        row = one_year_forward_curve.loc[closest_date]
        xs_fwd, ys_fwd = [], []
        for col in row.index:
            try:
                left, right = col.split('-')  # "0Y-1Y"
                t = int(left.replace('Y', ''))
                xs_fwd.append(t + 0.5)      # midpoint
                ys_fwd.append(float(row[col]))
            except Exception:
                continue

        if xs_fwd:
            fig.add_trace(go.Scatter(
                x=xs_fwd,
                y=ys_fwd,
                name='1Y Forwards (cross-section)',
                mode='lines+markers',
                line=dict(color='red', width=2,dash='dash')
            ))

    fig.update_layout(
        title='US Treasury Yield Curve',
        xaxis_title='Years to Maturity',
        yaxis_title='Yield / Forward (%)',
        template='plotly_dark',
        showlegend=True
    )

    return fig

def calculate_spreads(df):
    spreads = pd.DataFrame()
```

```
    if '2Y' in df.columns and '10Y' in df.columns:
        spreads['2s10s'] = df['10Y'] - df['2Y']
    if '3M' in df.columns and '10Y' in df.columns:
        spreads['3m10y'] = df['10Y'] - df['3M']
    if '5Y' in df.columns and '30Y' in df.columns:
        spreads['5s30s'] = df['30Y'] - df['5Y']
    return spreads


def plot_spreads(spreads):
    fig = go.Figure()
    for col in spreads.columns:
        if not spreads[col].isna().all():
            fig.add_trace(go.Scatter(
                x=spreads.index,
                y=spreads[col],
                name=col,
                line=dict(width=2)
            ))
    fig.add_hline(y=0, line_color='red', line_dash='dash')
    fig.update_layout(
        title='Treasury Yield Spreads',
        xaxis_title='Date',
        yaxis_title='Spread (bps)',
        template='plotly_dark'
    )
    return fig


def calculate_forward_rates(df):
    forwards = pd.DataFrame()
    maturities = {'3M': 0.25, '2Y': 2, '5Y': 5, '10Y': 10, '30Y': 30}
    for short_term, long_term in [('3M', '2Y'), ('2Y', '5Y'), ('5Y', '10Y'), ('10Y', '30Y')]:
        if short_term in df.columns and long_term in df.columns:
            r1 = maturities[short_term]
            r2 = maturities[long_term]
            try:
                forwards[f'{short_term}-{long_term}'] = (
                                        ((1 + df[long_term] / 100) ** r2 / (
```

```
                                        1 + df[short_term] / 100) ** r1) ** (
                                        1 / (r2 - r1)) - 1
                                ) * 100
            except Exception as e:
                print(f"Could not calculate forward rate for {short_term}-{long_term}: {e}")
    return forwards


def calculate_one_year_forward_curve(df: pd.DataFrame, maturities: dict, max_year: int = 30) -> pd.DataFrame:
    """
        Compute 1Y forward rates f(t,t+1) from spot yields (in %) for integer maturities t=0..max_year-1.
        Returns a DataFrame (index = dates, columns like '0Y-1Y', '1Y-2Y', ...), values in %.
    """
    cols_present = [c for c in df.columns if c in maturities]
    if not cols_present:
        return pd.DataFrame(index=df.index)

    maturity_years = np.array([float(maturities[c]) for c in cols_present], dtype=float)
    order = np.argsort(maturity_years)
    maturity_years = maturity_years[order]
    spot_df = df[[cols_present[i] for i in order]].astype(float)

    target_years = np.arange(0, max_year + 1, dtype=float)

    interp_list = []
    for date, row in spot_df.iterrows():
        yi = np.interp(x=target_years, xp=maturity_years, fp=row.values.astype(float))
        interp_list.append(pd.Series(yi, index=target_years, name=date))

    interp_spot = pd.DataFrame(interp_list)
    interp_spot.index = spot_df.index
    interp_spot[0.0] = 0.0
    interp_spot = interp_spot.reindex(sorted(interp_spot.columns), axis=1)

    fwd_cols = []
    fwd_data = {}
    for t in range(0, max_year):
        t0, t1 = float(t), float(t + 1)
        if t0 in interp_spot.columns and t1 in interp_spot.columns:
```

```
            z_t = interp_spot[t0] / 100.0
            z_tp1 = interp_spot[t1] / 100.0
            fwd = ((1.0 + z_tp1) ** (t + 1)) / ((1.0 + z_t) ** t) - 1.0
            col_name = f"{t}Y-{t + 1}Y"
            fwd_data[col_name] = fwd * 100.0
            fwd_cols.append(col_name)

    one_year_forward_curve = pd.DataFrame(fwd_data, index=interp_spot.index)[fwd_cols]
    return one_year_forward_curve



def plot_forward_rates(one_year_forward_curve):
    fig = go.Figure()
    for col in one_year_forward_curve.columns:
        if not one_year_forward_curve[col].isna().all():
            fig.add_trace(go.Scatter(
                x=one_year_forward_curve.index,
                y=one_year_forward_curve[col],
                name=col,
                line=dict(width=2)
            ))
    fig.update_layout(
        title='Implied Forward Rates',
        xaxis_title='Date',
        yaxis_title='Rate (%)',
        template='plotly_dark',
        showlegend=True
    )
    return fig



def plot_forward_rates(forwards):
    fig = go.Figure()
    for col in forwards.columns:
        if not forwards[col].isna().all():
            fig.add_trace(go.Scatter(
                x=forwards.index,
                y=forwards[col],
                name=col,
```

```
            line=dict(width=2)
        ))
    fig.update_layout(
        title='Implied Forward Rates',
        xaxis_title='Date',
        yaxis_title='Rate (%)',
        template='plotly_dark',
        showlegend=True
    )
    return fig


def main():
    st.set_page_config(layout="wide", page_title="Bond Yield Curve Analysis")
    st.title('📈 Treasuries Yield Curve Analysis Dashboard')

    col1, col2, col3 = st.columns([1, 2, 3])
    with col2:
        default_end_date = datetime.now()
        default_start_date = default_end_date - timedelta(days=365)
        date_range = st.date_input(
            'Select Date Range',
            [default_start_date, default_end_date],
            max_value=default_end_date
        )

    if len(date_range) == 2:
        start_date, end_date = date_range

        with st.spinner('Fetching yield data...'):
            # Your existing data fetch
            df = fetch_yield_data(start_date)

            # Existing forward rate calc (if you still use it)
            forwards = calculate_forward_rates(df)

            # 👉 Add new 1Y forward curve calc
            maturities = {
```

```
        '1M': 1 / 12, '3M': 0.25, '6M': 0.5, '1Y': 1, '2Y': 2,
        '3Y': 3, '5Y': 5, '7Y': 7, '10Y': 10, '20Y': 20, '30Y': 30
    }
    one_year_forward_curve = calculate_one_year_forward_curve(
        df=df,
        maturities=maturities,
        max_year=29
    )

if not df.empty:
    fig = plot_yield_curve(df, end_date, maturities)
    st.plotly_chart(fig, use_container_width=True)

        # Optionally, display data in an expander
    with st.expander("Show 1-Year Forward Curve Data"):
        st.dataframe(one_year_forward_curve.tail(10))

    spreads = calculate_spreads(df)
    if not spreads.empty:
        st.subheader('Yield Spread Analysis')
        col1, col2 = st.columns(2)
        with col1:
            st.plotly_chart(plot_spreads(spreads), use_container_width=True)
        with col2:
            current_spreads = spreads.iloc[-1]
            for spread, value in current_spreads.items():
                if pd.notna(value):
                    st.metric(f'{spread} Spread', f'{value:.2f} bps')

    # Forward rate section — no need to recalculate
    if not forwards.empty:
        st.subheader('Forward Rate Analysis')
        st.plotly_chart(plot_forward_rates(forwards), use_container_width=True)

    if '2s10s' in spreads.columns:
        current_2s10s = spreads['2s10s'].iloc[-1]
        if pd.notna(current_2s10s):
            st.subheader('Recession Probability Indicator')
            recession_prob = 1 / (1 + np.exp(-(-current_2s10s / 100)))
```

```
            st.metric('Recession Probability (Based on 2s10s spread)', f'{recession_prob:.1%}')


        st.subheader('Yield Curve Statistics')
        col1, col2 = st.columns(2)
        with col1:
            st.write('Summary Statistics')
            st.dataframe(df.describe())
        with col2:
            st.write('Correlation Matrix')
            st.dataframe(df.corr())
    else:
        st.error("Please select both start and end dates.")



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