Task 3

Gabriel Muñoz Luna - 105967744

The analysis of financial series requires working with precise data to represent the evolution in the market. One of the best forms to do this is via OHLCV format (Open, High, Low, Close, Volume), that permits to construct candle graphics to observe tendences and patrons.

In this document I will develop Task C.3 – Data Processing 2 that the objective is prepare the data in OHLCV format, apply temporal aggregation technics and generate static and interactive visualisation of data, comparing diary vs 5 days, to see the limits and advantages.

The data is from Yahoo Finance obtained via yfinance library, but also it could be local via CSV, this data contains columns like Open, High, Low, Close, Adj Close and Volume, that are the standard in financial series.

For the consistence, I use a function called “prepare_ohlc”, where the objective is clean, normalize ad secure that the data is valid to OHLC.

```python
def prepare_ohlc(df: pd.DataFrame, date_col: str = "date") -> pd.DataFrame:
    df = df.copy()
    df.columns = [c.strip().lower() for c in df.columns]

    if date_col not in df.columns:
        raise ValueError(f"Missing date column '{date_col}'")
    df[date_col] = pd.to_datetime(df[date_col], errors="coerce")
    df = df.dropna(subset=[date_col])

    # Normalize common names
    rename_map = {"adj close": "close"}
    for k, v in rename_map.items():
        if k in df.columns and v not in df.columns:
            df.rename(columns={k: v}, inplace=True)

    needed = ["open", "high", "low", "close"]
    maybe_volume = "volume" if "volume" in df.columns else None

    # Convert to numeric, cleaning common non-numeric chars
    df = _coerce_numeric_cols(df, needed + ([maybe_volume] if maybe_volume else []))

    # Eliminate rows with NaNs in needed columns
    drop_cols = needed + ([maybe_volume] if maybe_volume else [])
    df = df.dropna(subset=[c for c in drop_cols if c in df.columns])

    # Only keep needed columns + date_col, sort by date_col
    keep = [date_col] + needed + ([maybe_volume] if maybe_volume else [])
    df = df[keep].sort_values(date_col).reset_index(drop=True)
    return df
```

This creates a copy of the data frame and then

- Standardize names: where all the names are passed to lowercase and renamed to common names (Adj Close to Close).

- Valid dates: the column date, are converted to datetime type and eliminates invalid values.
- Cleaning of numeric data: the numbers are converted to numeric format, deleting invalid values. ($)
- Cleaning of NaN: if there is a NaN in the column is deleted.
- Final: only relevant columns (date, open, high, low, close, volume) are returned, sorted chronologically.

From this the data set are prepared to be utilized in the next steps

The resample is the process of regroup data in a temporal frequence to another, in this case the daily data is transform to wider intervals, like every 5 days, weekly or monthly. This permit to see smooth variations to analyse tendences more clearly.

```python
def resample_ohlcv(df: pd.DataFrame, freq: str = "5D", date_col: str = "date") -> pd.DataFrame:
    have_volume = "volume" in df.columns
    agg_map = {"open": "first", "high": "max", "low": "min", "close": "last"}
    if have_volume:
        agg_map["volume"] = "sum"
    g = (df.set_index(date_col)
           .resample(freq)
           .agg(agg_map)
           .dropna())
    return g.reset_index()
```

I use the function “resample_ohlcv” to do this process where:

- Index via date: where the column date is set as temporal index
- Resample: with .resampl(freq) the data are group in intervals (5D, W, M, etc.)
- Aggregation by columns:
    - Open: first value in interval
    - Hight: max value
    - Low: min value
    - Close: last value in interval
    - Volume: total sum of the volume in the interval
- Celanese: use .dropna() to eliminate incomplete intervals
- Final: Reset the index to return the ordered data frame

Once its prepared and regrouped in OHLCV format, it could be represented in candle charts. For this I use the library “mplfinance”, from matplotlib and is designed for specific finance data.

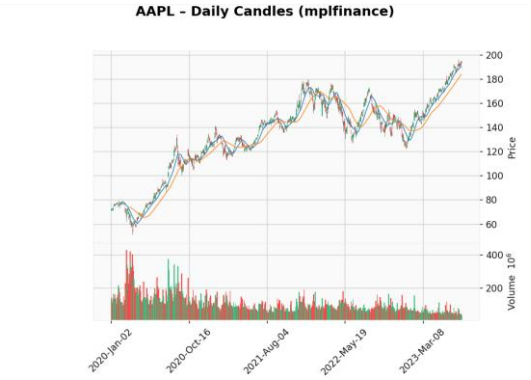
```python
def plot_candles_mpf(df: pd.DataFrame, date_col: str = "date",
                     title: str = "Candles", mav=(20, 50), volume=True):
    d2 = df.copy()
    # Make sure columns are numeric
    numeric_cols = ["open", "high", "low", "close"] + (["volume"] if "volume" in d2.columns else [])
    d2 = _coerce_numeric_cols(d2, numeric_cols)
    d2 = d2.dropna(subset=["open", "high", "low", "close"])  # mpf exige numéricos completos

    # mpf wait DatetimeIndex
    d2 = d2.set_index(date_col)
    # If date_col is not datetime, this will raise an error
    vol_flag = volume and ("volume" in d2.columns)
    mpf.plot(d2, type="candle", mav=mav, volume=vol_flag, title=title, style="yahoo")
```

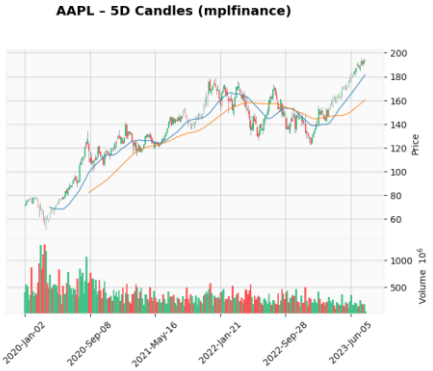
For this function:

- Mplfinance needs that the index to be DatetimeIndex, this is why the column of dates (date_col) is convert to the index before graphic
- The values of OHCL are used directly to construct the candles
    - The body of the candle represents the differences between high and low
    - The wicks represents high and low
- It could be added mobile meds to visualize tendences
- With the argument “volume, the volume is display in a sub graphic
- The “yahoo” style replicates the typical appearance og a financial graphic

An advantage of “mplfinance”is that generate quick and cleared graphs in a static format, adecuate to informs and analysis

Examples:



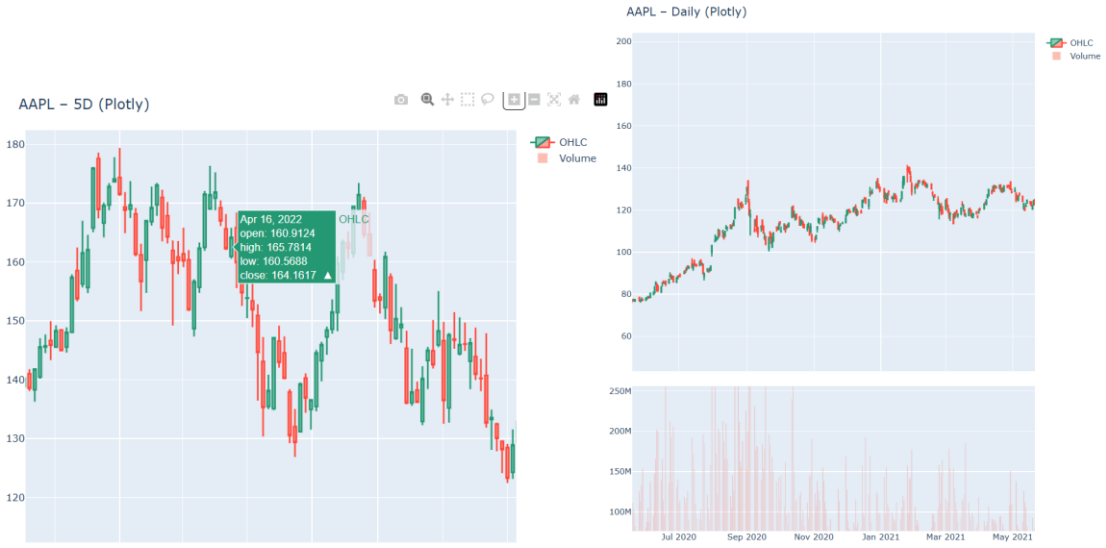


To complement the static view, I implement interactive view with “Plotly”. This one permit zoom, hover and other tooltips, that facilitate the exploration of patrons in different time. As difference as mplfinance. Plotly do not need Date time index, only with the date columne as X axis.

The function supports the sub graphic volume if exist, The OHLC is represent with candlestick and the volume with bar.

```python
def plot_candles_plotly(df, date_col="date", title="Candles (Plotly)", volume=True):
    d2 = df.copy()
    num = ["open","high","low","close"] + (["volume"] if "volume" in d2.columns else [])
    d2 = _coerce_numeric_cols(d2, num)
    d2[date_col] = pd.to_datetime(d2[date_col], errors="coerce")
    d2 = d2.dropna(subset=[date_col,"open","high","low","close"])

    has_vol = volume and ("volume" in d2.columns)
    if has_vol:
        fig = make_subplots(rows=2, cols=1, shared_xaxes=True, row_heights=[0.7,0.3], vertical_spacing=0.03)
        fig.add_trace(go.Candlestick(x=d2[date_col], open=d2["open"], high=d2["high"],
                                     low=d2["low"], close=d2["close"], name="OHLC"), row=1, col=1)
        fig.add_trace(go.Bar(x=d2[date_col], y=d2["volume"], name="Volume", opacity=0.4), row=2, col=1)
    else:
        fig = go.Figure(data=[go.Candlestick(x=d2[date_col], open=d2["open"], high=d2["high"],
                                             low=d2["low"], close=d2["close"], name="OHLC")])
    fig.update_layout(title=title, xaxis_rangeslider_visible=False, margin=dict(l=40,r=20,t=50,b=30))
    fig.show()
```

Example



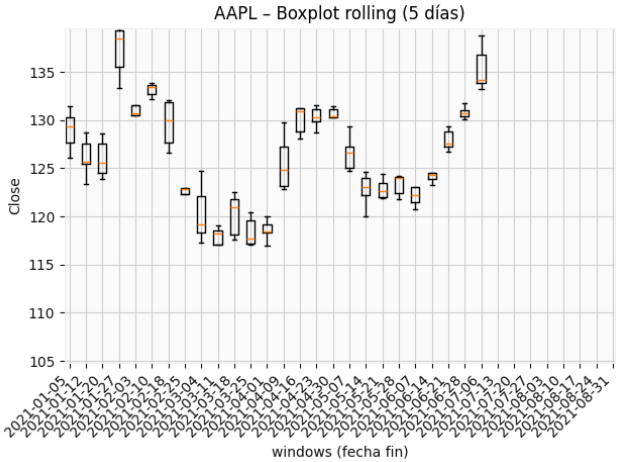
Beside the candlestick graph, I implement a function to see the close prices in boxplot form over a moving window of n consecutive days.

```python
def plot_boxplot_rolling(df, date_col="date", value_col="close", window=5, step=None, title=None):
    d2 = df.copy()
    d2[date_col] = pd.to_datetime(d2[date_col], errors="coerce")
    d2 = d2.dropna(subset=[date_col, value_col]).sort_values(date_col).reset_index(drop=True)

    vals = d2[value_col].to_numpy()
    dates = d2[date_col]                # <-- Serie de pandas (no numpy array)
    step = step or window               # ventanas no solapadas por defecto

    boxes, labels = [], []
    for start in range(0, len(vals)-window+1, step):
        end = start + window
        boxes.append(vals[start:end])
        # etiqueta = fin de ventana, ahora sí tiene .strftime
        labels.append(dates.iloc[end-1].strftime("%Y-%m-%d"))

    if not boxes:
        raise ValueError("No hay suficientes datos para la ventana elegida.")

    plt.figure()
    plt.boxplot(boxes, showfliers=False)
    plt.xticks(ticks=range(1, len(labels)+1), labels=labels, rotation=45, ha="right")
    plt.title(title or f"Boxplot rolling de {value_col} (ventana={window})")
    plt.xlabel("Ventanas (fecha fin)")
    plt.ylabel(value_col.capitalize())
    plt.tight_layout()
    plt.show()
```

The function “plot_boxplot_rolling” generates a blox for each interval days, showing:

- Median: central of price
- Interquartile range (IQR): dispersion of values
- Whiskers: minimums and maximums within the range
- Outliers: extreme values (in this case, disabled with showfliers=False for clarity).

In the graphic, every box correspond to the close prices of a block, this permit a simple see of price dispersion in the intervals and detect high volatile periods.



Comparative

The analysis of financial series can vary significantly depending on the time interval selected for the construction of the candles:

- Daily: each candle represents a operative days, proportionating g the maximum of details, util for short term but at the same time this include a lot of noise (little fluctuations that are no clear)
- 5 days: each candle resumes 5 days of prices, this time frame softens the variations and is more used in the technical analysis, because is easily to identify patterns.

LIbraryes

- Mplfinance: offers high-quality static visualizations, ideal for reports, but lacks interactivity.
- Plotly: offers interactivity (zoom, hover, hide series), but may require more resources and configuration, and must open in a browser rather than being embedded directly into a PDF.

References

Pandas (2024). *Time series / date functionality*. pandas.pydata.org. https://pandas.pydata.org/docs/user_guide/timeseries.html

PythonFinTech (2023). *Financial charts with mplfinance*. https://pythonfintech.com/

Plotly (2024). *Candlestick charts in Python*. plotly.com. https://plotly.com/python/candlestick-charts/