Instructions

1. Rename this file with your name and date of submission E.g., john_doe_yyyymmdd.ipynb
2. The csv files include ohlc and volume data from 2024-03-18 to 2024-06-14. For the week of **May 27th** (2024-05-27 to 2024-05-31), identify important prices or ranges for that week. Define "important" and provide an explanation. You can use data outside of this range to test hypotheses.
3. Analyze the data and create a module to identify this important price.
    - (Optional) Create a ranking system to score probability or price importance.
4. Feel free to use any resources available.
5. Provide a write up below or in a separate document to explain your thought process, hypotheses, and any other interesting points you observed through your analysis.

Keep in mind anomalous times including 08:30 EST due to high impact news. 10:00 AM etc. Use this calendar.

Evaluation Metrics
We will assess your submission based on the following three categories:

1. Novelty of idea/approach
2. Analytical ability and scientific reasoning
3. Coding skill

Table Schema

- **ts_event**: time in EST
- **open**: price at start of time interval
- **high**: max price during time interval
- **low**: min price during time interval
- **close**: last price of time interval
- **volume**: total volume traded (A + B + N)
- **volume_A**: total volume traded on A side
- **volume_B**: total volume traded on B side
- **volume_N**: total volume traded on N side

In [39]:
```
import pandas as pd
import numpy as np
```

In [165…
```
df_1D = pd.read_csv('1D.csv')
df_60min = pd.read_csv('60min.csv')
df_5min = pd.read_csv('5min.csv')
df_1min = pd.read_csv('1min.csv')
df_10s = pd.read_csv('10s.csv')
```

I will begin by creating a function to filter data for a specific date range.

In [166…
```
df_60min['ts_event'] = pd.to_datetime(df_60min['ts_event'])

start_date = "2024-05-27"
end_date = "2024-05-31 23:59:59"

def filter_data(df, start_date, end_date):
    df['ts_event'] = pd.to_datetime(df['ts_event'])
    return df[(df['ts_event'] >= start_date) & (df['ts_event'] <= end_date)]

# I will create a data frame df_week that only considers the data from the weel
df_week = filter_data(df_60min, start_date, end_date)
```

Defining the Weekly Range

By the values of the prices in the ohlc collumn, I highly suspect that this data is for the ESM2024 futures contract. I will check the highest and lowest point of the weekly high to low and confirm this with the data on Trading View.

In [10]:
```
highest_price_row = df_week.loc[df_week['high'].idxmax()]
highest_price = highest_price_row['high']
highest_price_time = highest_price_row['ts_event']

lowest_price_row = df_week.loc[df_week['low'].idxmin()]
lowest_price = lowest_price_row['low']
lowest_price_time = lowest_price_row['ts_event']

print(f"Highest price: {highest_price} reached at {highest_price_time}")
print(f"Lowest price: {lowest_price} reached at {lowest_price_time}")
```

```
Highest price: 5339.0 reached at 2024-05-28 04:00:00-04:00
Lowest price: 5205.5 reached at 2024-05-31 12:00:00-04:00
```

We can see that the weekly candle traded in a range of 133.5 points from high to low. Indeed, this corresponds with the data available in Trading View, so I can confirm that this data is for ES.

Economic Calendar

Before we start any analysis, let's check the economic calendar. Since we know we are looking at ES, I will filter the economic calendar to only include USD news that is high impact. We can see that Monday is a bank holiday, meaning that it is less likely that we will get high quality opportunity to trade (depending on the strategy we use) since there is much less volume during the day. In addition, we can see 10am Consumer Confidence on

Tuesday, and GDP / Inital Jobless Claims getting released at 8:30 on Thursday. In general, we want to avoid trading before high impact news releases, as price tends to consolidate and be more lethargic in anticipation for the injection of volatility at those times.

Defining Important Prices

We can define 4 different types of price ranges that we can classify as "important".

1. **High Volume Areas**: Prices where there is a significant amount of trading volume. This often indicates institutional interest or strong participation by market participants. We are interested in these places because it is where price moves more efficiently towards where we anticipate it to go.
2. **Support and Resistance Levels**: Prices where the market consistently uses to bounce back up or back down from. These can also be viewed as places where swing highs / swing lows were formed.
3. **Time**: We can also consider prices that are traded to during hours where the market is volatile to be more important.
4. **VWAP**: I will classify price ranges that are farther away from the weekly VWAP to be "important" since they are more likely to be "overbough" or "oversold".

The goal is to create an algorithm that assigns a score to price bins depending on these factors. In order to do so, I will create four helper functions: one for each of these metrics. The goal of each helper function is to output a normalized score from 0 to 1 of their corresponding metric for each price bin. At the end, my rank_prices function will compute a weighted sum of all these scores and output the most "important" price bins in order based on these factors.

1. High Volume Areas

This function computes the volume score for each price bin. The score is based on the trading volume at each price bin, normalized between 0 and 1. Higher volume indicates more trading activity and thus increases the score. The score is scaled so that the price bins with the highest volume get a score of 1, while bins with lower volume get a proportionally lower score.

In [250…

```
def calculate_volume_score(df, price_range=5):
    df = df.copy()

    # Create price bins
    df['price_bin'] = (df['close'] // price_range) * price_range

    # Calculate total volume for each price bin
    volume_by_bin = df.groupby('price_bin')['volume'].sum().reset_index()

    # Normalize
    max_volume = volume_by_bin['volume'].max()
    volume_by_bin['volume_score'] = volume_by_bin['volume'] / max_volume
```

```
    df = df.merge(volume_by_bin[['price_bin', 'volume_score']], on='price_bin'

    return df
```

Here's an example of this function giving a volume score for each price bin:

In [249…
```
volume_score_example = calculate_volume_score(df_week, price_range=5).drop_dup
print(volume_score_example[['price_bin', 'volume_score']].head())
```

```
    price_bin  volume_score
0      5315.0      0.138862
1      5320.0      0.153214
10     5325.0      0.465505
11     5330.0      0.018676
23     5335.0      0.024100
```

2. Support and Resistance Levels

This function identifies price reversals (swing highs and lows) and assigns a reversal score to each price bin. Price bins with more frequent reversals are given higher scores, as these points indicate key support or resistance levels in the market. The score is normalized between 0 and 1, with higher scores indicating price bins with more reversals.

In [253…
```
def find_price_reversals(df, window=1, price_range=5):
    df = df.copy()

    # Identify swing highs and swing lows
    df['swing_high'] = df['high'][(df['high'] > df['high'].shift(window)) & (d
    df['swing_low'] = df['low'][(df['low'] < df['low'].shift(window)) & (df['l
    df['reversal_price'] = df['swing_high'].fillna(df['swing_low'])

    # Create price bins
    df['price_bin'] = (df['reversal_price'] // price_range) * price_range

    # Group by price_bin and count the number of reversals in each bin
    reversal_scores = df.groupby('price_bin').agg({
        'reversal_price': 'count'
    }).reset_index()

    # Normalize
    max_reversals = reversal_scores['reversal_price'].max()
    reversal_scores['reversal_score'] = (reversal_scores['reversal_price'] / ma

    return reversal_scores[['price_bin', 'reversal_score']]
```

Here's an example of this function giving a reversal score for each price bin:

In [252…
```
price_reversal_scores = find_price_reversals(df_week, window=1, price_range=5)
print(price_reversal_scores.head())
```

```
   price_bin  reversal_score
0     5205.0             0.2
1     5230.0             0.4
2     5235.0             0.6
3     5245.0             0.6
4     5250.0             0.4
```

3. Time

Thirdly, I will classify important price levels as ones that have been traded to during key times of interest. These will be determined based on the historic volatility of the market at every given time of the day. In order to do this, first I display a data frame below that measures the volatility of each hourly candle using the historical data provided based on the standard deviation of the hourly candle range.

In [174…

```
# Assign a score for the hours that have the best volatility based on all the (
df_60min['range'] = df_60min['high'] - df_60min['low']  # High-low range as a p
hourly_volatility = df_60min[df_60min['ts_event'] < "2024-05-27"].groupby(df_60
hourly_volatility.columns = ['hour', 'volatility']
hourly_volatility
```

Out[174]:

	hour	volatility
0	0	1.468255
1	1	2.635023
2	2	3.666430
3	3	3.243747
4	4	3.514154
5	5	3.018782
6	6	3.765564
7	7	4.678464
8	8	17.025102
9	9	7.635810
10	10	10.275672
11	11	6.934395
12	12	7.015142
13	13	10.646746
14	14	13.433284
15	15	12.068705
16	16	9.803273
17	18	4.006505
18	19	3.039598
19	20	3.721605
20	21	7.216425
21	22	4.253968
22	23	3.560116

The function below generates a time-based volatility score for each price bin. It looks at the volatility of the market during different hours and assigns higher scores to price bins that occur during historically volatile periods. This score is normalized between 0 and 1, with higher values representing price bins that coincide with higher volatility hours. For example, if a price zone is traded inside of at 10:00am, it will receive a much higher score compared to a price level that is traded at 16:00 when the NYSE is closed.

In [173…

```
def calculate_time_score(df, hourly_volatility):
    df = df.copy()
    df['time_score'] = 0

    # Assign volatility score based on the hour of the event
    df['hour'] = pd.to_datetime(df['ts_event']).dt.hour

    # Volatility score for each hour
    df = df.merge(hourly_volatility, on='hour', how='left')
```

```
    df['time_score'] = (df['volatility'] / df['volatility'].max())

    return df
```

4. VWAP

Finally, I will create a function to calculate the VWAP of data set given the Close prices and the Volume. To classify important prices this week, I will use the VWAP price for the whole week.

In [171…
```
def calculate_vwap(df):
    return (df['close'] * df['volume']).sum() / df['volume'].sum()

vwap_price = calculate_vwap(df_week)
print(f"VWAP Price for the week: {vwap_price}")
```

```
VWAP Price for the week: 5278.01147782906
```

Final Price Ranking

Finally, I defined the rank_prices algorithm below using the previous helper functions. It is designed to assign a total score from 0 to 1 to each price bin based on the following factors in descending order of importance: volume, number of price reversals, time volatility, and distance to VWAP. I assigned the weights of these factors to be 40%, 30%, 20%, and 10%, respectively.

In [257…
```
def rank_prices(df, vwap_price, hourly_volatility, price_range=5):
    df = df.copy()  # Copy to avoid warnings

    # Create price bins
    df['price_bin'] = (df['close'] // price_range) * price_range

    # Calculate volume score
    df = calculate_volume_score(df, price_range = price_range)  # Volume weigh

    # Distance to VWAP score
    df['distance_to_vwap'] = np.abs(df['price_bin'] - vwap_price)
    df['vwap_score'] = (df['distance_to_vwap'] / df['distance_to_vwap'].max())

    # Volatility score
    df = calculate_volatility_score(df, hourly_volatility)

    # Reversal points score
    reversal_scores = find_price_reversals(df, window=1, price_range=price_ran

    # Merge the reversal scores into the main DataFrame
    df = df.merge(reversal_scores[['price_bin', 'reversal_score']], on='price_

    # Total score combining all factors
    df['total_score'] = 0.4 * df['volume_score'] + 0.3 * df['reversal_score'] +

    # Sort by total score
    df = df.groupby('price_bin').agg({'total_score': 'max'}).reset_index()
```

```
    df = df.sort_values('total_score', ascending=False)

    return df
```

This is the top ranking of the price bins for the week of May 27th 2024 with a score from 0 to 1 for each:

In [256…

```
ranked_prices = rank_prices(df_week, vwap_price, hourly_volatility, price_rang
print(ranked_prices.head())
```

```
    price_bin  total_score
12     5295.0     0.868736
18     5325.0     0.520773
5      5250.0     0.493168
1      5230.0     0.486962
2      5235.0     0.464685
```