Overview

template notebook using F# (and C# for backend libs) to research intraday stock patterns using minute OHLCV data.

In [ ]:
```
#!import "build_stock_research_dep.ps1"
```

In [1]:
```
#!import "import_libs.fsx"
```

In [ ]:
```
#!import "shared_imports_and_setup_fsharp.ipynb"
```

In [3]:
```
#!import "shared_clients_fsharp.ipynb"
```

In [4]:
```
if shared_clients_fsharp_canary <> 1 then
    failwith "shared_clients_fsharp did not load, make sure all referenced librarie
```

get list of symbol/date pairs of stocks up highest price % change between [prior trade day reg hours close at 16:00, current day 09:25], between dates: [2023-08-01, 2023-08-31]

In [18]:
```
let getSymbolUniverseIndividualTiRequestDto = GetSymbolUniverseIndividualTiRequestD
    FilterName = "ChangeFromClosePct",
    BiggestFirst = true,
    Time = LocalTime(9,25),
    StartDate = LocalDate(2023,8,1),
    EndDate = LocalDate(2023,8,31)
)

let getSymbolUniverseRequestDto = GetSymbolUniverseRequestDto(
    GetSymbolUniverseIndividualRequests = ResizeArray.ofList [
        getSymbolUniverseIndividualTiRequestDto
    ]
)
```

In [19]:
```
let runTaskSync t =
    t
    |> Async.AwaitTask
    |> Async.RunSynchronously
```

In [9]:
```
let symbolUniverseService = container.Resolve<SymbolUniverseService.SymbolUniverseS
```

In [21]:
```
let symbolUniverseResponse =
        let t = symbolUniverseService.GetSymbolUniverseAsync(getSymbolUniverseReque
        t |> runTaskSync

let symbolUniverseEntries =
    symbolUniverseResponse.SymbolsUniverseCollections.First().SymbolUniverseEntries
```

```
symbolUniverseEntries.Count
```

Out[21]: 1768

In [22]:
```
let symbolUniverseEntriesToKeep =
    symbolUniverseEntries
        // .Skip(0)
        // .Take(1);

// symbolUniverseEntriesToKeep
```

In [60]:
```
let symbolDatePairs =
    symbolUniverseEntriesToKeep
    |> Seq.map (fun x -> SymbolDatePair(x.Symbol, x.StartTrackingTime.GetNyDate()))
    |> List.ofSeq

symbolDatePairs |> Seq.head
```

Out[60]:

▼ *(ABVC, 2023-08-01)*

Symbol	ABVC
Date	▶ *Tuesday, August 1, 2023*

In [63]:
```
let dateAndSymbolsFirstDay =
    symbolDatePairs
    |> Seq.groupBy (fun x -> x.Date)
    |> Seq.head
```

In [25]:
```
symbolDatePairs  |> Seq.map (fun x -> x.Symbol) |> Seq.distinct |> Seq.length
```

Out[25]: 1218

In [26]:
```
let getOhlcPriceSeriesRequestDto  =
    MarketDataShared.GetOhlcPriceSeriesRequestDto(
        SamplingFreq = TradeServicesSharedDotNet.Models1.SamplingFreq.Minute,
        SymbolDatePairs = (symbolDatePairs |> ResizeArray.ofSeq),
        DownloadMissing = false
    )
```

In [27]:
```
let marketDataService = container.Resolve<MarketDataService.MarketDataService>()
```

In [28]:
```
let ohlcPriceReturnMsgs =
    let t =
        marketDataService.GetOhlcPriceSeriesAsync(getOhlcPriceSeriesRequestDto, Thr
            .ToListAsync()
            .AsTask()
    t |> runTaskSync

ohlcPriceReturnMsgs.Count
```

Out[28]: 1602

In [29]:
```
type Ohlc = {Time:DateTime; Open:double; High:double; Low:double; Close:double; Vol

let unwrapOhlcEntry (ohlcEntry: OhlcEntry) =
    let ohlcVals = ohlcEntry.OhlcEntryVals
    { Ohlc.Time = ohlcEntry.Time.AsDateTimeInNyTz();
     Open = double ohlcVals.Open;
     High = double ohlcVals.High;
     Low = double ohlcVals.Low;
     Close = double ohlcVals.Close;
     Volume = ohlcVals.Volume }

let unwrapAllOhlcEntries ohlcEntries =
    ohlcEntries |> Seq.map unwrapOhlcEntry

// ohlcPriceReturnMsgs[0].OhlcEntries |> unwrapAllOhlcEntries
```

In [30]:
```
let dateSymbolsWithOhlcs =
    ohlcPriceReturnMsgs
    |> Seq.map (fun x ->
        (x.SymbolDateSamplingFreqPair.Date, x.SymbolDateSamplingFreqPair.Symbol),
         x.OhlcEntries |> unwrapAllOhlcEntries |> Seq.toList)
```

In [ ]:
```
dateSymbolsWithOhlcs |> Seq.head
```

In [32]:
```
let (symbolDate, ohlcs) = dateSymbolsWithOhlcs |> Seq.head
let ohlcFrame = ohlcs |> Frame.ofRecords
ohlcFrame
ohlcFrame |> Frame.indexRowsUsing (fun s -> s.Get("Time")) |> Frame.dropCol "Time"
```

Out[32]:

A frame: 827 x 5

	Open	High	Low	Close	Volume
	System.Double	System.Double	System.Double	System.Double	System.UInt32
8/1/2023 4:00:00 AM	0.18	0.1876	0.1618	0.185	75586
8/1/2023 4:01:00 AM	0.1813	0.1813	0.1749	0.18	85697
8/1/2023 4:02:00 AM	0.175	0.1822	0.175	0.1776	17392
8/1/2023 4:03:00 AM	0.1787	0.1799	0.175	0.1799	58020
8/1/2023 4:04:00 AM	0.1795	0.18	0.1782	0.1799	53747
8/1/2023 4:05:00 AM	0.1786	0.1786	0.1748	0.175	15621
8/1/2023 4:06:00 AM	0.175	0.1787	0.1748	0.1787	17879
8/1/2023 4:07:00 AM	0.1749	0.1794	0.1748	0.176	15176
8/1/2023 4:08:00 AM	0.1788	0.1794	0.1748	0.1779	26482
8/1/2023 4:09:00 AM	0.1708	0.1765	0.169	0.1745	28031
8/1/2023 4:10:00 AM	0.1747	0.175	0.1699	0.1748	23698
8/1/2023 4:11:00 AM	0.1705	0.1749	0.1705	0.1749	13905
8/1/2023 4:12:00 AM	0.174	0.1788	0.174	0.1766	19042
8/1/2023 4:13:00	0.1755	0.1768	0.1749	0.175	19251

	Open	High	Low	Close	Volume
	System.Double	System.Double	System.Double	System.Double	System.UInt32
AM					
8/1/2023 4:14:00 AM	0.1749	0.1751	0.174	0.1741	23806
8/1/2023 4:15:00 AM	0.1739	0.174	0.1713	0.172	15327
8/1/2023 4:16:00 AM	0.1714	0.173	0.1713	0.173	2438
8/1/2023 4:17:00 AM	0.1755	0.177	0.1713	0.176	11329
8/1/2023 4:18:00 AM	0.1755	0.176	0.1743	0.176	2087
8/1/2023 4:19:00 AM	0.1767	0.177	0.1759	0.1759	7367
...	...	...	...	...	...

...with 807 additional rows

In [33]:
```
let frame_per_date_symbol_pair = new Dictionary<_, _>()

for (symbolDate, ohlcs) in dateSymbolsWithOhlcs do
    let ohlcFrame =
        ohlcs
        |> Frame.ofRecords
        |> Frame.indexRowsUsing (fun s -> s.GetAs<DateTime>("Time"))
        |> Frame.dropCol "Time"
    frame_per_date_symbol_pair.Add(symbolDate, ohlcFrame)
```

In [34]:
```
frame_per_date_symbol_pair |> Seq.head
```

Out[34]:

▼ *[(Tuesday, August 1, 2023, AGRI), Deedle.Frame`2[System.DateTime,System.String]]*

Key ▶ *(Tuesday, August 1, 2023,*

Value A frame: 827 x 5

	Open	High	Low	Close	V
	System.Double	System.Double	System.Double	System.Double	System.
8/1/2023 4:00:00 AM	0.18	0.1876	0.1618	0.185	
8/1/2023 4:01:00 AM	0.1813	0.1813	0.1749	0.18	
8/1/2023 4:02:00 AM	0.175	0.1822	0.175	0.1776	
8/1/2023 4:03:00 AM	0.1787	0.1799	0.175	0.1799	
8/1/2023 4:04:00 AM	0.1795	0.18	0.1782	0.1799	
8/1/2023 4:05:00 AM	0.1786	0.1786	0.1748	0.175	
8/1/2023 4:06:00 AM	0.175	0.1787	0.1748	0.1787	
8/1/2023 4:07:00 AM	0.1749	0.1794	0.1748	0.176	
8/1/2023 4:08:00 AM	0.1788	0.1794	0.1748	0.1779	
8/1/2023 4:09:00 AM	0.1708	0.1765	0.169	0.1745	
8/1/2023 4:10:00 AM	0.1747	0.175	0.1699	0.1748	
8/1/2023 4:11:00 AM	0.1705	0.1749	0.1705	0.1749	

	Open	High	Low	Close	
	System.Double	System.Double	System.Double	System.Double	System
8/1/2023 4:12:00 AM	0.174	0.1788	0.174	0.1766	
8/1/2023 4:13:00 AM	0.1755	0.1768	0.1749	0.175	
8/1/2023 4:14:00 AM	0.1749	0.1751	0.174	0.1741	
8/1/2023 4:15:00 AM	0.1739	0.174	0.1713	0.172	
8/1/2023 4:16:00 AM	0.1714	0.173	0.1713	0.173	
8/1/2023 4:17:00 AM	0.1755	0.177	0.1713	0.176	
8/1/2023 4:18:00 AM	0.1755	0.176	0.1743	0.176	
8/1/2023 4:19:00 AM	0.1767	0.177	0.1759	0.1759	
...	...	...	...	...	

...with 807 additional rows

In [35]:
```
let displayLength f =
    let tmp = f |> Seq.length
    display($"length: {tmp}")

frame_per_date_symbol_pair |> displayLength
```

```
length: 1602
```

In [36]:
```
let toTuple (source: KeyValuePair<'a, 'b>) =
    Tuple.Create(source.Key, source.Value)
```

In [37]:
```
let getPremarketVolume (ohlcFrame: Frame<_,_>) =
    try
        ohlcFrame["Volume"]
        |> getBetweenTimesHelper None (TimeOnly(9,30)|> Some)
        |> Series.scanValues (fun acc v -> acc + v) 0.0
```

```
        |> Series.lastValue
        |> Nullable
    with
        | :? System.IndexOutOfRangeException -> Nullable()

let getPremarketAvgVolumePerMinute (ohlcFrame: Frame<_,_>) =
    try
        let filteredFrame =
            ohlcFrame["Volume"]
            |> getBetweenTimesHelper None (TimeOnly(9,30)|> Some)
        let meanFromNon0TradeVol =
            filteredFrame
            |> Stats.mean
        let minutesWithNon0Vol = float filteredFrame.KeyCount
        let totPreMarketMinutes = 330.
        let meanFromNon0TradeVolNonNan =
            if System.Double.IsNaN(meanFromNon0TradeVol) then
                0.0
            else
                meanFromNon0TradeVol
        let meanPerMinuteInclMinWith0Vol = meanFromNon0TradeVolNonNan * minutesWith
        meanPerMinuteInclMinWith0Vol |> Nullable
    with
        | :? System.IndexOutOfRangeException -> 0.0 |> Nullable

let getMarketOpenPrice f =
    try
        let filteredFrame = f |> getBetweenTimesHelperFrame (TimeOnly(9,30) |> Some
        filteredFrame["Open"] |> Series.firstValue |> Nullable
    with
         | :? System.IndexOutOfRangeException -> Nullable()

let getPreMarketHigh f =
    try
        let filteredFrame =
            f |> getBetweenTimesHelperFrame None (TimeOnly(9,30) |> Some)
        filteredFrame["High"] |> Stats.max |> Nullable
    with
         | :? System.IndexOutOfRangeException -> Nullable()


let getPreMarketOpen (f: Frame<_,_>) =
    let firstTime = f.RowKeys |> Seq.head |> TimeOnly.FromDateTime

    if firstTime >= TimeOnly(9,30) then
        Nullable()
    else f["Open"] |> Series.firstValue |> Nullable

let getMarketClosePrice f =
    try
        let filteredFrame =
            f |> getBetweenTimesHelperFrame (TimeOnly(9,30) |> Some) (TimeOnly(16,0
        filteredFrame["Close"] |> Series.lastValue |> Nullable
    with
         | :? System.IndexOutOfRangeException -> Nullable()
```

```
let getRegMarketHigh f =
    try
        let filteredFrame =
            f |> getBetweenTimesHelperFrame (TimeOnly(9,30) |> Some) (TimeOnly(16,0
        filteredFrame["High"] |> Stats.max |> Nullable
    with
         | :? System.IndexOutOfRangeException -> Nullable()
```

In [38]:
```
let (x,y) = frame_per_date_symbol_pair |> Seq.skip 0 |>  Seq.head |> toTuple
[getPremarketAvgVolumePerMinute] |> List.map (fun f -> f y)
```

Out[38]:

▼ *[37461.90303030303]*

HeadOrDefault	37461.90303030303
TailOrNull	▶ *[ ]*
Head	37461.90303030303
Tail	▶ *[ ]*
(values)	[37461.90303030303]

In [39]:
```
let ohlcKvTuples =
    frame_per_date_symbol_pair
    |> Seq.map toTuple

let keys =
    ohlcKvTuples
    |> Seq.map fst

let calcOnOhlcs func =
    let mapped =
        ohlcKvTuples
        |> Seq.map (fun (_, ohlcs) ->
             (func ohlcs))
    Series<_,_>(keys, mapped)


let preMarketVolume = calcOnOhlcs getPremarketVolume
let premarketAvgVolumePerMinute = calcOnOhlcs getPremarketAvgVolumePerMinute
let preMarketOpenPrices = calcOnOhlcs getPreMarketOpen
let preMarketHighPrices = calcOnOhlcs getPreMarketHigh
let marketOpenPrices = calcOnOhlcs getMarketOpenPrice
let regMarketHighPrices = calcOnOhlcs getRegMarketHigh
let marketClosePrices = calcOnOhlcs getMarketClosePrice

let df =
    ["preMarketVolume" => preMarketVolume;
    "preMarketAvgVolumePerMinute" => premarketAvgVolumePerMinute;
    "preMarketOpenPrices" => preMarketOpenPrices;
    "preMarketHigh" => preMarketHighPrices;
    "marketOpenPrices" => marketOpenPrices;
    "regHoursHigh" => regMarketHighPrices;
    "marketClosePrices" => marketClosePrices]
    |> frame
```

```
let df_no_nulls = df |> Frame.filterRowValues (fun row -> row.ValueCount = row.KeyC

let calcReturnSeries col1 col2 =
    df_no_nulls?(col1) / df_no_nulls?(col2) - 1.

let gapSeries = calcReturnSeries "preMarketHigh" "preMarketOpenPrices"
df_no_nulls.AddColumn ("gapPct", gapSeries)

let preMarketHighToOpenReturnSeries = calcReturnSeries "preMarketHigh" "marketOpenP
df_no_nulls.AddColumn ("preMarketHighToOpenReturn", preMarketHighToOpenReturnSeries

let regHoursHighReturnSeries = calcReturnSeries "regHoursHigh" "marketOpenPrices"
df_no_nulls.AddColumn ("regHoursHighReturn", regHoursHighReturnSeries)

let eodReturnSeries = calcReturnSeries "marketClosePrices" "marketOpenPrices"
df_no_nulls.AddColumn ("eodReturn", eodReturnSeries)

df_no_nulls |> Frame.take 3
```

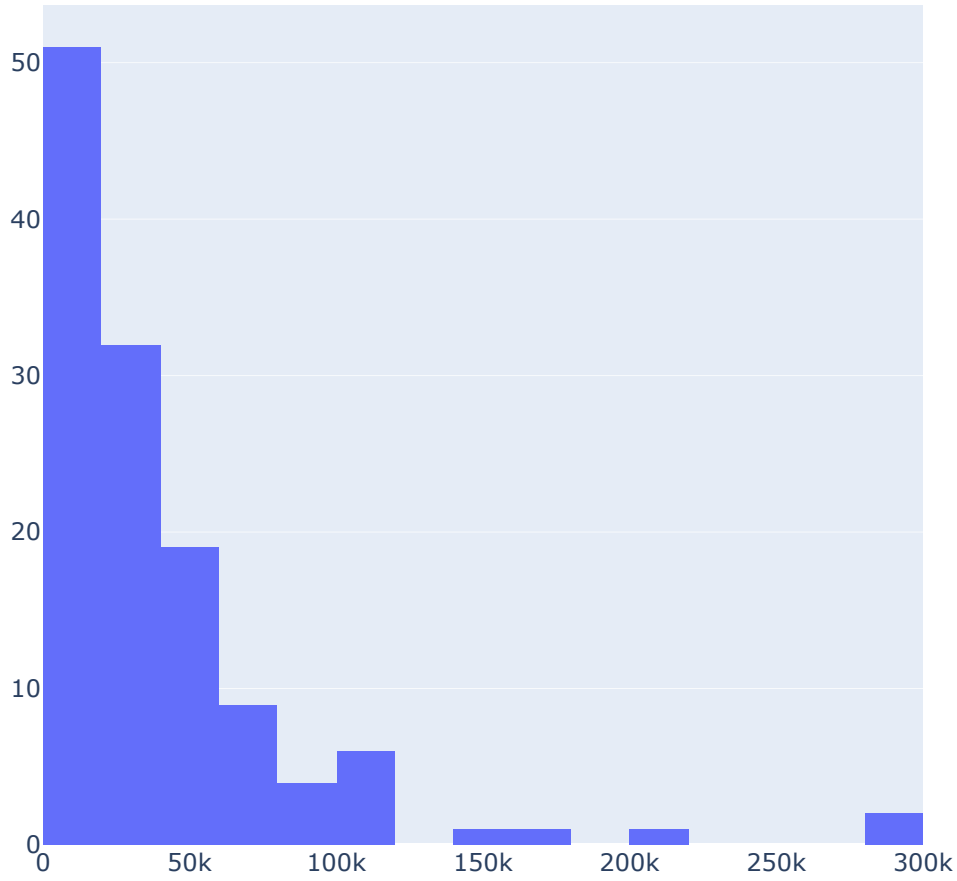
Out[39]:

	preMarketVolume	preMarketAvgVolumePerMinute	preMarketOpenPrices	preMar
	System.Double	System.Double	System.Double	System
(Tuesday, August 1, 2023, AGRI)	12362428	37461.90303030303	0.18	
(Tuesday, August 1, 2023, AMSC)	6148170	18630.81818181818	11	
(Tuesday, August 1, 2023, ANET)	218905	663.3484848484849	178.45	

In [40]:

```
let x = df_no_nulls["preMarketAvgVolumePerMinute"] |> Series.filterValues (fun x ->
Chart.Histogram(x |> Series.values)
```

Out[40]:



In [41]:
```
// gap %s in buckets

df_no_nulls?("gapPct")
    |> Series.filter (fun _ v -> v >= 0.1)
    |> Series.groupBy (fun k v -> int(v * 100. / 10.) * 10)
    |> Series.mapValues Stats.count
    |> Series.sortByKey
```

Out[41]:

A series: 20 values. Key type: System.Int32

Keys	10	20	30	40	50	60	70	80	90	100	110	120	130	140	170	...
Values	222	93	30	26	20	15	11	8	3	6	4	4	2	1	1	...

In [42]:
```
let rowContainsVal v row =
    row
    |> Series.values
```

```
    |> List.ofSeq
    |> List.exists ((=) v)
```

In [43]:
```
let getRet thresh (row: ObjectSeries<_>) =
    let regHoursHighReturn = row?("regHoursHighReturn")
    let eodReturns = row?("eodReturn")
    if regHoursHighReturn >= thresh || eodReturns >= thresh then
        thresh
    else
        eodReturns

let stratReturns =
    df_no_nulls |> Frame.mapRowValues (getRet 0.2)
    // |> Series.filterValues (fun v -> v > 0.)
    // |> Stats.median

stratReturns * -1. |>  Series.foldValues (fun acc v -> acc * (1. + v)) 0.01
```

Out[43]: 2.769341702265871

In [44]:
```
// count per high return  - e.g. 0.1 key = returns in range [0.1,0.2)
df_no_nulls["regHoursHighReturn"]
|> Series.groupBy (fun k v -> float((int(v * 10.)) % 10) / 10.)
|> Series.mapValues Stats.count
|> Series.sortByKey
```

Out[44]:

A series: 9 values. Key type: System.Double

Keys	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
Values	989	197	66	31	12	8	6	5	3

In [45]:
```
let listToTuple l =
    let l' = List.toArray l
    let types = l' |> Array.map (fun o -> o.GetType())
    let tupleType = Microsoft.FSharp.Reflection.FSharpType.MakeTupleType types
    Microsoft.FSharp.Reflection.FSharpValue.MakeTuple (l' , tupleType)

let tupleToList t =
    if Microsoft.FSharp.Reflection.FSharpType.IsTuple(t.GetType())
        then Some (Microsoft.FSharp.Reflection.FSharpValue.GetTupleFields t |> Arra
        else None

let tupleToList2 t =
    Microsoft.FSharp.Reflection.FSharpValue.GetTupleFields t |> Array.toList

let unzip sequence =
    let (lstA, lstB) =
        Seq.foldBack (fun (a,b) (accA, accB) ->
            a::accA, b::accB) sequence ([],[])
    (Seq.ofList lstA, Seq.ofList lstB)
```

In [46]:
```
let extractXy frame =
    frame
        |> Frame.rows
```

```
        |> Series.mapValues (Series.values >> List.ofSeq >> listToTuple)
        |> Series.values
        |> Seq.map unbox<float * float>
        |> unzip

let plotPoints frame =
    let (x, y) = frame |> extractXy
    Chart.Scatter (x,y, mode=StyleParam.Mode.Markers)
```

In [47]:
```
df_no_nulls
    // |> Frame.filterRowValues (fun r -> r?("preMarketVolume") > 10_000.)
    |> Frame.filterRowValues (fun r -> r?("preMarketAvgVolumePerMinute") >= 10_000.
```

Out[47]:

	preMarketVolume	preMarketAvgVolumePerMinute	preMarketOpenPrices	preM
	System.Double	System.Double	System.Double	Syst
(Tuesday, August 1, 2023, AGRI)	12362428	37461.90303030303	0.18	
(Tuesday, August 1, 2023, AMSC)	6148170	18630.81818181818	11	
(Tuesday, August 1, 2023, MGAM)	3949029	11966.754545454545	0.7898	
(Tuesday, August 1, 2023, NKLA)	38389689	116332.39090909091	2.82	
(Tuesday, August 1, 2023, QBTS)	8074759	24468.96666666667	2.25	
(Tuesday, August 1, 2023, QUBT)	5825787	17653.9	1.45	
(Tuesday, August 1, 2023, TTOO)	13037531	39507.6696969697	0.1519	
(Tuesday, August 1, 2023, TUP)	11160436	33819.50303030303	4.64	
(Wednesday, August 2, 2023, LIDR)	13913940	42163.454545454544	0.5717	
(Wednesday, August 2, 2023, RBT)	6988137	21176.172727272726	1.49	
(Wednesday, August 2, 2023, CRKN)	57685650	174805	0.067	
(Wednesday, August 2, 2023, AHI)	4591147	13912.566666666668	3	
(Wednesday, August 2,	15788273	47843.25151515152	0.284	

	preMarketVolume	preMarketAvgVolumePerMinute	preMarketOpenPrices	preM
	System.Double	System.Double	System.Double	Syst
2023, WAVD)				
(Thursday, August 3, 2023, DUO)	17531739	53126.48181818182	0.2201	
(Thursday, August 3, 2023, SYTA)	33801833	102429.79696969697	0.0537	
(Thursday, August 3, 2023, TTOO)	67591640	204823.15151515152	0.325	
(Thursday, August 3, 2023, BMRA)	4532782	13735.70303030303	1.53	
(Friday, August 4, 2023, AMZN)	5602170	16976.272727272728	139.91	
(Friday, August 4, 2023, FUBO)	8617242	26112.854545454546	3.37	
(Friday, August 4, 2023, MF)	24458734	74117.37575757576	0.68	
...	...	...	...	

...with 106 additional rows

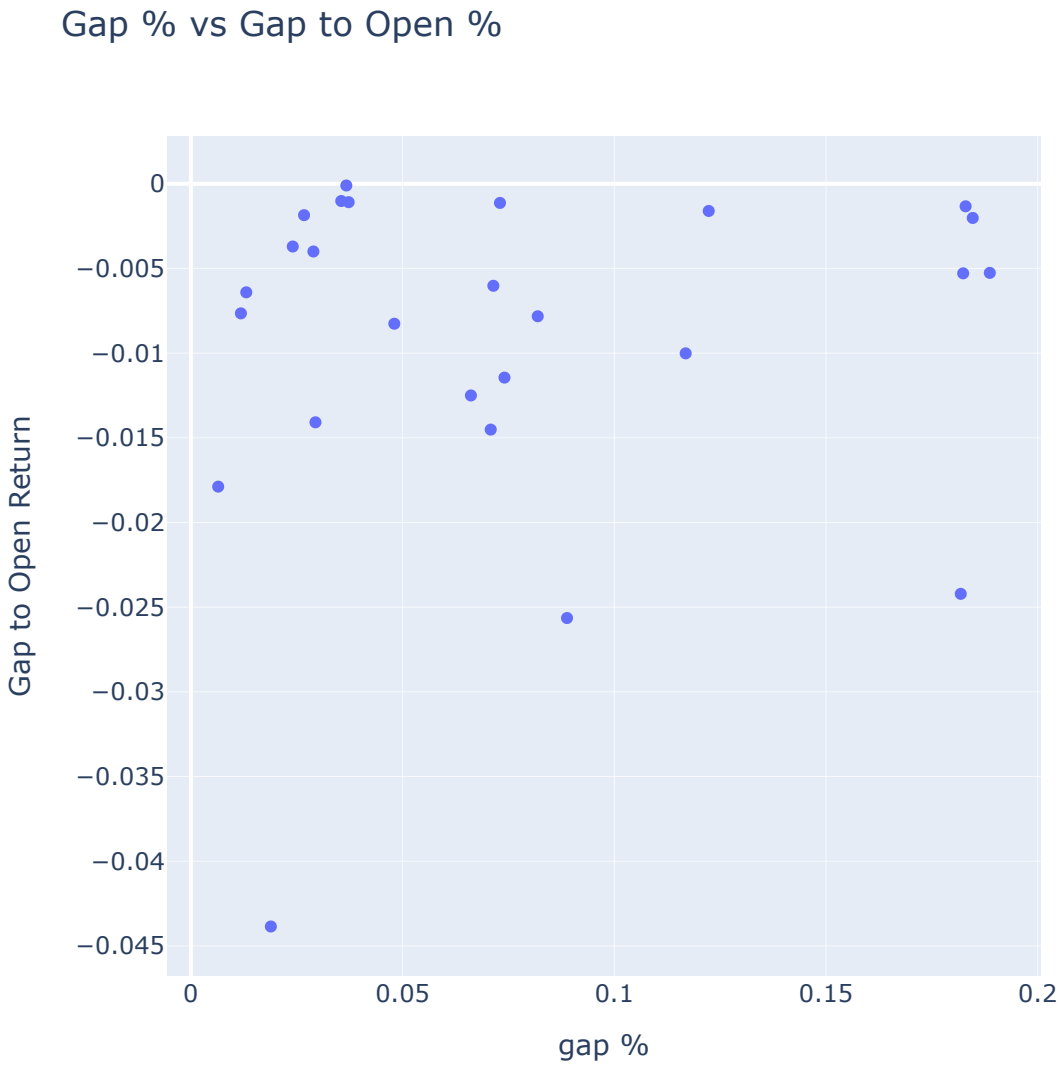
In [48]:
```
let filteredFrame =
    let filtered =
        df_no_nulls
            // |> Frame.filterRowValues (fun r -> r?("preMarketVolume") > 10_000.)
            |> Frame.filterRowValues (fun r -> r?("marketOpenPrices") >= 1.)
            |> Frame.filterRowValues (fun r -> r?("preMarketAvgVolumePerMinute") >=
            |> Frame.filterRowValues (fun r -> r?("gapPct") <> 0)
            |> Frame.filterRowValues (fun r -> r?("preMarketHighToOpenReturn") < -0
    // filtered.Columns[["gapPct"; "eodReturn"; "preMarketHighToOpenReturn"]]
    filtered
        |> Frame.sortRowsWith "eodReturn" (-)

let gapEodFrame = filteredFrame.Columns[["gapPct"; "eodReturn"]]
let gapVsGapToOpen = filteredFrame.Columns[["gapPct"; "preMarketHighToOpenReturn"]]
```

```
filteredFrame.RowCount |> sprintf "rows: %i" |> display

filteredFrame |> Frame.take 5 |> display

gapEodFrame
    |> plotPoints
    |> Chart.withXAxisStyle ("gap %")
    |> Chart.withYAxisStyle ("EOD returns")
    |> Chart.withTitle ("Gap % vs EOD returns")
    |> display

gapVsGapToOpen
    |> plotPoints
    |> Chart.withXAxisStyle ("gap %")
    |> Chart.withYAxisStyle ("Gap to Open Return")
    |> Chart.withTitle ("Gap % vs Gap to Open %")
    |> display
```

```
rows: 26
```

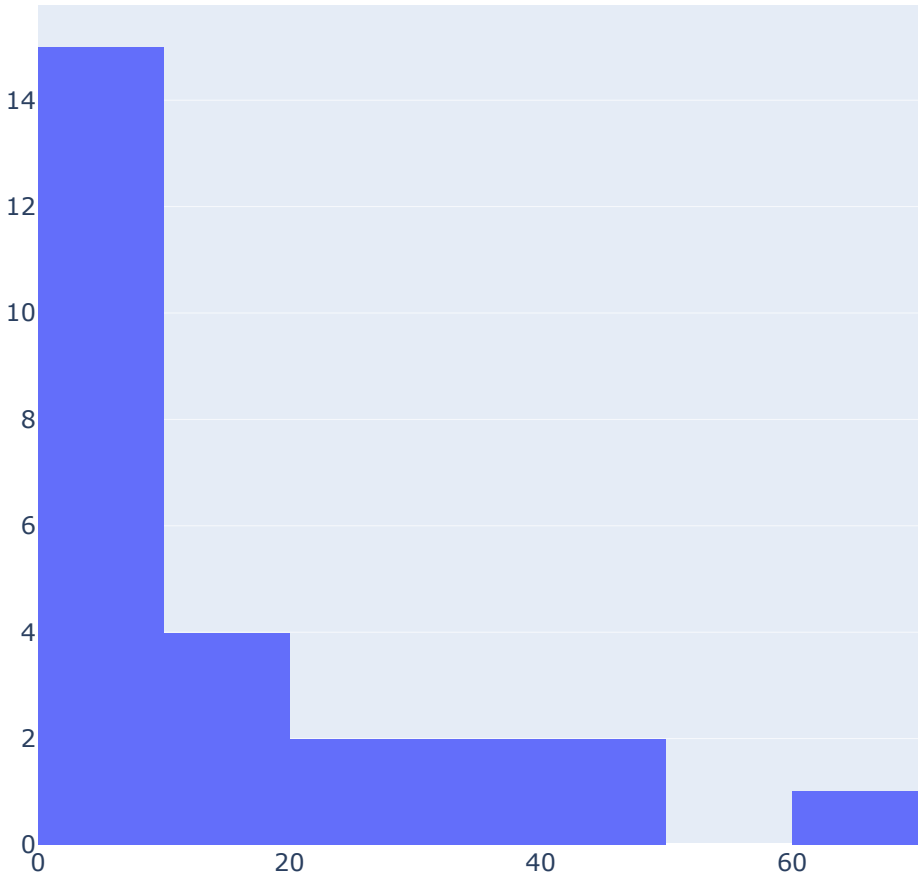
	preMarketVolume	preMarketAvgVolumePerMinute	preMarketOpenPrices	preMa
	System.Double	System.Double	System.Double	Syster
(Tuesday, August 1, 2023, DBVT)	20964	63.527272727272724	1.54	
(Wednesday, August 30, 2023, CONN)	36489	110.57272727272728	3.41	
(Tuesday, August 29, 2023, GSHD)	10341	31.336363636363636	66.19	
(Tuesday, August 29, 2023, BIG)	375830	1138.878787878788	6.34	
(Monday, August 28, 2023, BNOX)	44862	135.94545454545454	1.59	


Gap % vs EOD returns
EOD returns
0.15
0.1
0.05
0
−0.05
−0.1
−0.15
−0.2
0
0.05
0.1
0.15
0.2
gap %



In [49]:
```
filteredFrame["marketOpenPrices"]
|> Series.values
|> Chart.Histogram
```

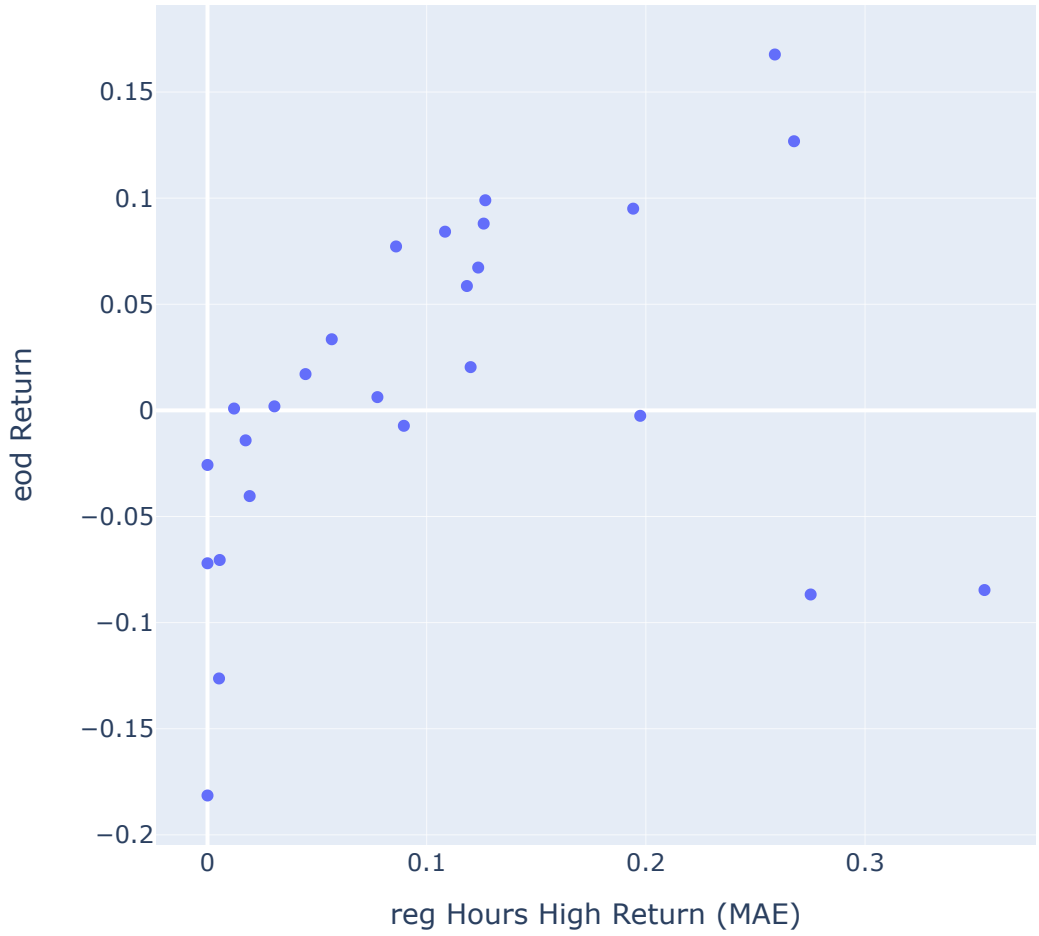
Out[49]:



In [50]:

```
filteredFrame.Columns[["regHoursHighReturn"; "eodReturn"]]
|> plotPoints
|> Chart.withXAxisStyle ("reg Hours High Return (MAE)")
|> Chart.withYAxisStyle ("eod Return")
```

Out[50]:



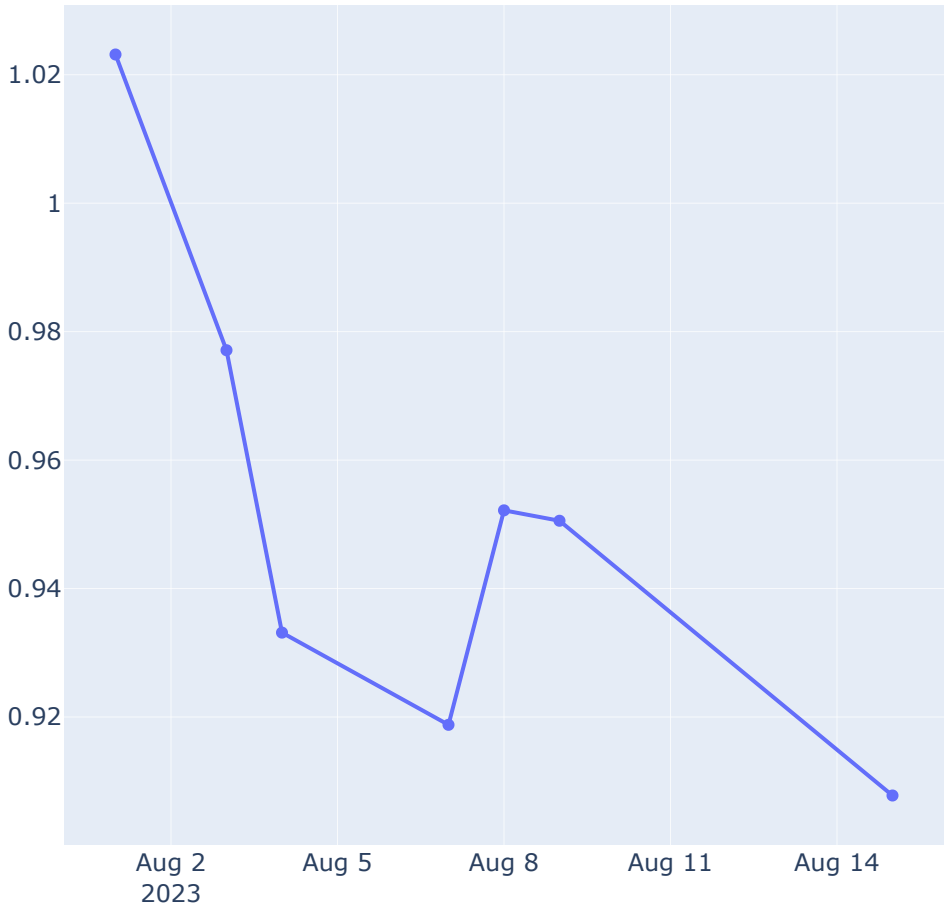
In [51]:
```
let groupByF k v =
    let (d, s) = k
    d
```

In [52]:
```
let truncate count (series:Series<'K, 'T>) =
    if count < 0 then
        invalidArg "count" "Must be greater than zero."
    let until =
        if count > series.KeyCount then
            series.KeyCount - 1
        else
            count
    series.GetAddressRange(RangeRestriction.Start(int64 until))
```

In [53]:
```
let retPerRow (r: ObjectSeries<_>) =
    let regHoursHighReturn = r.GetAs<float>("regHoursHighReturn")
    let cutOff = 0.05
    if regHoursHighReturn >= cutOff then
```

```
        -1. * cutOff
    else
        -1. * r.GetAs<float>("eodReturn")

let shortingReturnsWithStopLoss =
    filteredFrame |> Frame.mapRowValues retPerRow

let avgReturnPerDay =
    shortingReturnsWithStopLoss
        |> Series.groupInto groupByF (fun dtSymbol y -> y |> truncate 4 |> Stats.me
        |> Series.sortByKey

let cumReturn = avgReturnPerDay |> Series.scanValues (fun acc v -> acc * (1. + 0.9
let y = cumReturn.Values
let x = cumReturn.Keys |> Seq.map (fun x -> x.ToDateTimeUnspecified())

Chart.Scatter(x , y, StyleParam.Mode.Lines_Markers)
```

Out[53]:



In [54]:
```
let plotFrame (dict: Dictionary<_, Frame<_,_>>) key =
    let (date, sym) = key
    let f1 = dict[key]
    let closeSeries = f1["Close"]
    Chart.Scatter(closeSeries.Keys, closeSeries.Values, StyleParam.Mode.Lines_Marke
    |> Chart.withLineStyle(Shape=StyleParam.Shape.Hv)
    |> Chart.withTitle $"{sym} {date}"
```

In [58]:
```
let skipN = 1
let k = filteredFrame.RowKeys |> Seq.skip skipN |> Seq.head
k
k |> plotFrame frame_per_date_symbol_pair
```

Out[58]: