Data Analysis

#1. Essential Types of Financial Data

- Fundamental data is extremely regularised and low frequency. It encompasses information that can be found in regulatory filings and business analytics. It is mostly accounting data, reported quarterly.
  - The data can be reported with a lapse
  - The data is often backfilled or restated
- Market data includes all trading activity that takes place in an exchange or trading venue.
  - The data is not trivial to process
  - The data generated on a daily basis has a decent size for research
- Analytics data is kind of derivative data, which the original source could be, fundamental, market data, etc.
  - The data could be very costly
  - The data can be biased or opaque
- Alternative data is produced by individuals (social media, news, etc.), business processes (transactions, corporate data, etc.), and sensors (satellites, weather, etc.)
  - The data is primary information and truly unique
  - The data is expensive and hard to process

#2. Bars

In order to apply ML algorithm on the unstructured data, we need to parse it, extract valuable information from it, and store those extractions in a regularised format.

- Time bars are obtained by sampling information at fixed time intervals, e.g., once every week. The information collected usually includes: Timestamp, Volume-weighted average price, open, close, high, low, volume
  - Time bars are the most popular among practitioners and academics
  - But time bars should be avoided for two reasons: a) markets do not process information at a constant time interval b) time-sampled series often exhibit poor statistical properties, like correlation, heteroscedasticity, and non-normality of returns
- Tick bars is extracted each time a pre-defined number of transactions takes place, e.g. 1000 ticks
  - Tick-sampled data allows us to achieves returns closer to IID Normal
  - Tick-sampled data is sensitive to outliers
- Volume bars are sampling every time a pre-defined amount of security's units (shares, contracts, etc.)
  - Volume bars data provide better statistical properties then sampling by tick
  - The data gives good representation of market microstructure, e.g. price and volume
- Dollar bars are formed by sampling an observation every time a pre-defined market value is exchanged
  - Dollar bars are robust to outstanding shares actions

#3. Sampling Features

One reason for sampling features from a structured dataset is to reduce the amount of data used to fit the ML algorithm. The operation is also referred to as downsampling

- Reduction sampling
  - Linspace and uniform sampling
- Event-based sampling
  - CUMSUM filter is designed to detect a shift in the mean value of a measured quantity away from a target value

#3. Sampling Features (cont.)

SNIPPET 2.4 THE SYMMETRIC CUSUM FILTER

```
def getTEvents(gRaw,h):
    tEvents,sPos,sNeg=[],0,0
    diff=gRaw.diff()
    for i in diff.index[1:]:
        sPos,sNeg=max(0,sPos+diff.loc[i]),min(0,sNeg+diff.loc[i])
        if sNeg<-h:
            sNeg=0;tEvents.append(i)
        elif sPos>h:
            sPos=0;tEvents.append(i)
    return pd.DatetimeIndex(tEvents)
```

#4. Triple-Barrier Method

Triple-barrier method labels an observation according to the first barrier touched out three barriers.

- Two horizontal barriers are defined by profit-taking and stop-loss limits, which are a dynamic function of estimated volatility
  - If the upper barrier is touched first, we label the observation as a 1 or the return. If the lower barrier is touched first, we label the observation as a -1 or the return.
- The vertical barrier is defined in terms of number of bars elapsed since the position was taken (an expiration limit)
  - If the vertical barrier is touched, we label the observation 0 or the return

#4. Triple-Barrier Method (cont.)

SNIPPET 3.1 DAILY VOLATILITY ESTIMATES

```
def getDailyVol(close,span0=100):
    # daily vol, reindexed to close
    df0=close.index.searchsorted(close.index-pd.Timedelta(days=1))
    df0=df0[df0>0]
    df0=pd.Series(close.index[df0-1], index=close.index[close.shape[0]-df0.shape[0]:])
    df0=close.loc[df0.index]/close.loc[df0.values].values-1 # daily returns
    df0=df0.ewm(span=span0).std()
    return df0
```

SNIPPET 3.2 TRIPLE-BARRIER LABELING METHOD

```
def applyPtSlOnT1(close,events,ptSl,molecule):
    # apply stop loss/profit taking, if it takes place before t1 (end of event)
    events_=events.loc[molecule]
    out=events_[['t1']].copy(deep=True)

    if ptSl[0]>0:pt=ptSl[0]*events_['trgt']
    else:pt=pd.Series(index=events.index) # NaNs
    if ptSl[1]>0:sl=-ptSl[1]*events_['trgt']
    else:sl=pd.Series(index=events.index) # NaNs
    for loc,t1 in events_['t1'].fillna(close.index[-1]).iteritems():
        df0=close[loc:t1] # path prices
        df0=(df0/close[loc]-1)*events_.at[loc,'side'] # path returns
        out.loc[loc,'sl']=df0[df0<sl[loc]].index.min() # earliest stop loss.
        out.loc[loc,'pt']=df0[df0>pt[loc]].index.min() # earliest profit taking.
    return out
```

#4. Triple-Barrier Method (cont.)

Let us denote a barrier configuration by the triplet [pt, sl, t1], where 0 means that the barrier is inactive and a 1 means that a barrier is active. Three useful configurations are:

- [1, 1, 1]: this is the standard setup, where we define three barrier exit conditions. We would like to realise a profit, but we have a maximum tolerance for losses and a holding period
- [0, 1, 1]: in this setup, we would like to exit after a number of bars, unless we are stopped-out
- [1, 1, 0]: here we would like to take a profit as long as we are not stopped-out. This is somewhat unrealistic in that we are willing to hold the position for as long as it takes

#4. Triple-Barrier Method (cont.)

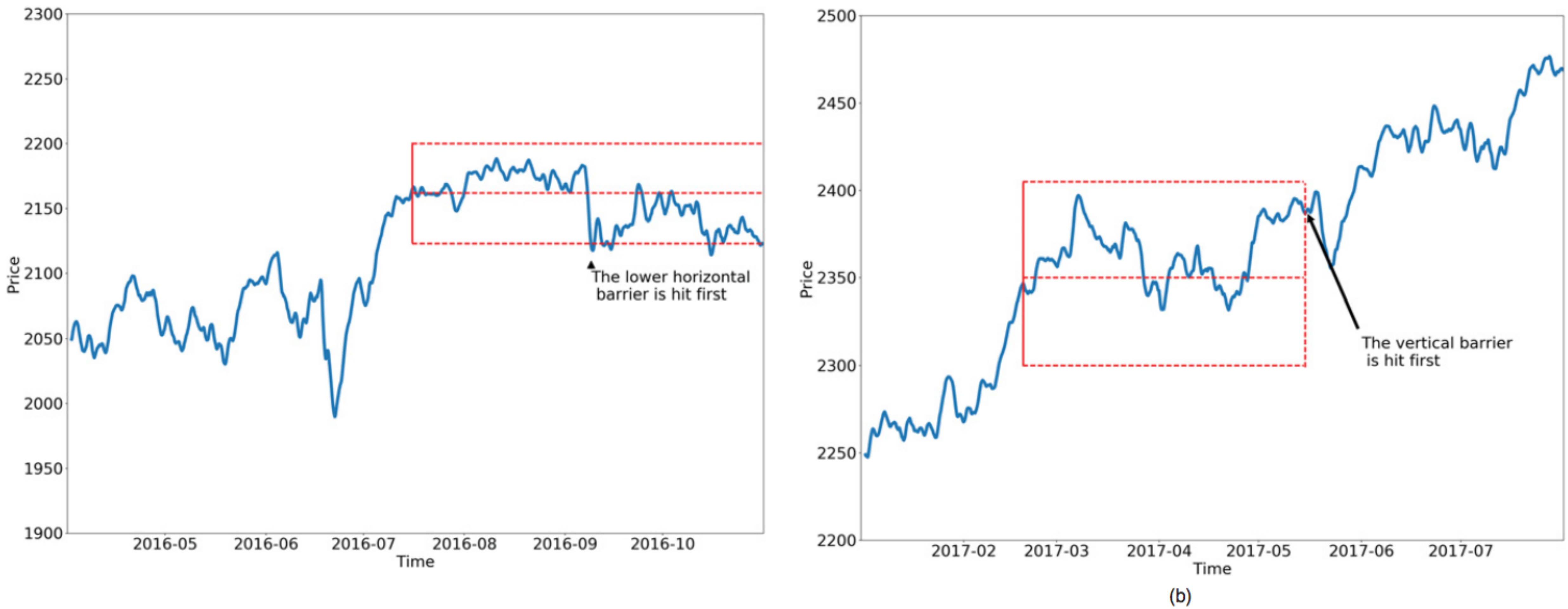

FIGURE 3.1 Two alternative configurations of the triple-barrier method

#4. Triple-Barrier Method (cont.)

SNIPPET 3.3 GETTING THE TIME OF FIRST TOUCH

```
def getEvents(close,tEvents,ptSl,trgt,minRet,numThreads,t1=False):
    #1) get target
    trgt=trgt.loc[tEvents]
    trgt=trgt[trgt>minRet] # minRet
    #2) get t1 (max holding period)
    if t1 is False:t1=pd.Series(pd.NaT,index=tEvents)
    #3) form events object, apply stop loss on t1
    side_=pd.Series(1.,index=trgt.index)
    events=pd.concat({'t1':t1,'trgt':trgt,'side':side_}, \
        axis=1).dropna(subset=['trgt'])
    df0=mpPandasObj(func=applyPtSlOnT1,pdObj=('molecule',events.index), \
        numThreads=numThreads,close=close,events=events,ptSl=[ptSl,ptSl])
    events['t1']=df0.dropna(how='all').min(axis=1) # pd.min ignores nan
    events=events.drop('side',axis=1)
    return events
```

- `close`: A pandas series of prices.
- `tEvents`: The pandas timeindex containing the timestamps that will seed every triple barrier. These are the timestamps selected by the sampling procedures discussed in Chapter 2, Section 2.5.
- `ptSl`: A non-negative float that sets the width of the two barriers. A 0 value means that the respective horizontal barrier (profit taking and/or stop loss) will be disabled.
- `t1`: A pandas series with the timestamps of the vertical barriers. We pass a `False` when we want to disable vertical barriers.
- `trgt`: A pandas series of targets, expressed in terms of absolute returns.
- `minRet`: The minimum target return required for running a triple barrier search.
- `numThreads`: The number of threads concurrently used by the function.

#5. Meta-Labelling

Suppose that you have a model for setting the side of the bet (long or short). When side is not None, meta-labelling is triggered. We build an ML model to effectively discriminate between profit taking and stop loss. The algorithm is trained to decide whether to take a bet or pass, a purely binary prediction.

- Binary classification problems present a trade-off type-I error (FT: false positives) and type-II error (false negatives)
- Meta-labelling is particularly helpful when you want to achieve higher F1-scores
  - First, we build a model to achieve high recall (even the precision is not high)
  - Second, we correct for the low precision by meta-labelling. The method will increase your F1-score by filtering out the false positives

#6. Bagging Classifiers and Uniqueness

we assigned a label y to an observed feature X_i , where y_i was a function of price bars that occurred over an interval $[t_{i,0}, t_{i,1}]$. When $t_{i,1} > t_{j,0}$ and $i < j$, then y_i and y_j will both depend on a common return $r_{tj,0\ ,\min\{ti,1\ ,tj,1\}}$, that is, the return over the $[t_{j,0},\min\{t_{i,1},t_{j,1}\}]$. The implication is that the series of labels, $\{y_i\}_{i=1,\ldots,I}$, are not IID whenever there is an overlap between any two consecutive outcomes, interval
$\exists i \mid t_{i,1} > t_{i+1,0}$.

There are several ways to attack the problem of non-IID labels

- We can restrict outcome's horizon to eliminate overlaps
  - In this case, the sampling frequency will be limited by the horizon used to determine the outcome
- Utilize the average uniqueness to reduce the undue influence of outcomes that contain redundant information
  - in this way, the in-bag observations will not be sampled at a frequency much higher than their uniqueness
- Sequencial bootstrap
  - Draws are mode according to a changing probability that controls for redundancy

#7. Stationary vs. Memory Dilemma

It is common in finance to find non-stationary time series. What makes these series non-stationary is the presence of memory, i.e., a long history of previous levels that shift the series' mean over time.
In order to perform inferential analysis, researchers need to work with invariant processes, such as returns on prices. These data transformations make the series stationary, at the expense of removing all memory from the original series.

- This arithmetic series consists of a dot product, *d* is a real positive number

$$\tilde{X}_t = \sum_{k=0}^{\infty} \omega_k X_{t-k}$$

with weights ω

$$\omega = \left\{ 1, -d, \frac{d(d-1)}{2!}, -\frac{d(d-1)(d-2)}{3!}, \ldots, (-1)^k \prod_{i=0}^{k-1} \frac{d-i}{k!}, \ldots \right\}$$

and values X

$$X = \{X_t, X_{t-1}, X_{t-2}, X_{t-3}, \ldots, X_{t-k}, \ldots\}$$

#7. Stationary vs. Memory Dilemma (cont.)

SNIPPET 5.3 THE NEW FIXED-WIDTH WINDOW FRACDIFF METHOD

```
def fracDiff_FFD(series,d,thres=1e-5):
    '''
    Constant width window (new solution)
    Note 1: thres determines the cut-off weight for the window
    Note 2: d can be any positive fractional, not necessarily bounded [0,1].
    '''
    #1) Compute weights for the longest series
    w=getWeights_FFD(d,thres)
    width=len(w)-1
    #2) Apply weights to values
    df={}
    for name in series.columns:
        seriesF,df_=series[[name]].fillna(method='ffill').dropna(),pd.Series()
        for iloc1 in range(width,seriesF.shape[0]):
            loc0,loc1=seriesF.index[iloc1-width],seriesF.index[iloc1]
            if not np.isfinite(series.loc[loc1,name]):continue # exclude NAs
            df_[loc1]=np.dot(w.T,seriesF.loc[loc0:loc1])[0,0]
        df[name]=df_.copy(deep=True)
    df=pd.concat(df,axis=1)
    return df
```

#8. Stationary with Maximum Memory Preservation

Consider a series $\{X_t\}_{t=1,\dots,T}$. Applying the fixed-width window fracdiff (FFD) method on this series, we can compute the minimum coefficient d* such that the resulting fractionally differentiated series $\{\tilde{X}_t\}_{t=l^*,\dots,T}$ is stationary. This coefficient d* quantifies the amount of memory that needs to be removed to achieve stationarity.

the figure illustrates the concept. On the right y-axis, it plots the ADF statistic computed on E-mini S&P 500 futures log-prices. On the x-axis, it display d value used to generate the series on which the ADF statistic was computed. The original series has an ADF statistic of -0.3387, while the returns series has an ADF statistic of -46.9114. At a 95% confidence level, the test's critical value is -2.8623. The ADF statistic crosses that threshold in the vicinity of d=0.35

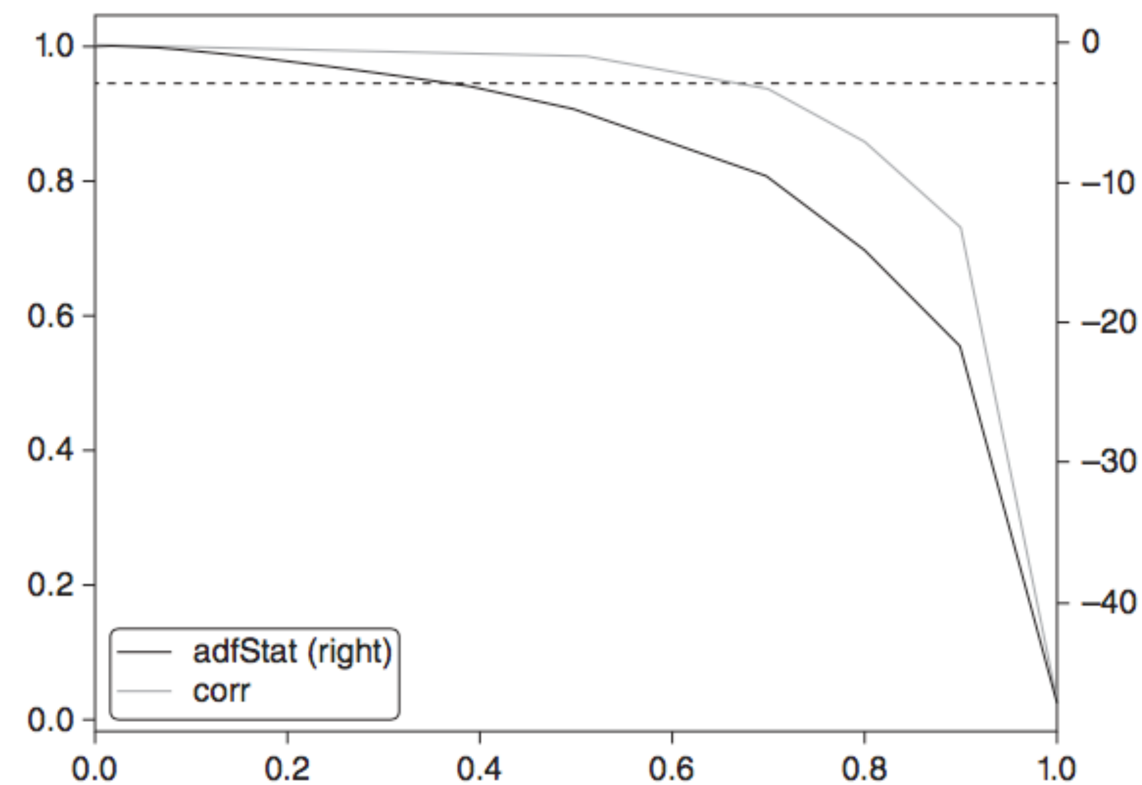

FIGURE 5.5 ADF statistic as a function of *d*, on E-mini S&P 500 futures log-prices