## **vn.py社区精选15 - 量化策略评估的显微镜：逐笔对冲统计（下）**

原创 KeKe VNPY  2019-12-13

收录于话题

#vn.py社区精选

25个

**通用型逐笔成交统计**

逐笔成交统计想用通用化，难点在于去限定一次完整开平交易的开始点和结束点，抽象来说就是寻找特殊的断点对所有成交记录进行划分。

**断点的选择**

而在算法状态机控制中，我们可以知道数字0是一个非常有用的评判标准，即我们构建一列数据，让它数值在完全平仓后变成0，就知道真正的平仓时间。

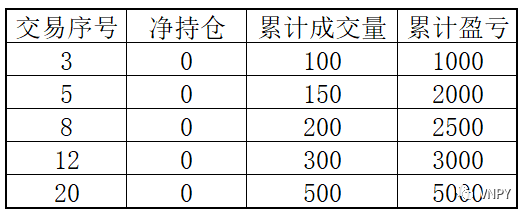
在实践中，累计净持仓恰恰好符合这个标准，我们把多头仓位设为”+”，空头仓位设为“-”，得到如下表的【方向持仓】，对【方向持仓】进行累计得到【净持仓】。

这样，我们基于【净持仓】为0可以得到每次开平交易的结束点。而该结束点为成交记录的断点。



**使用断点划分成交记录**

为了简单演示，下面我们只显示【净持仓】（列）为0的成交信息（行），如下表所示，一共发生了5开完整的开平仓交易。每笔交易的结束点对应的交易序号分别为3、5、8、12、20。这5个结束点即为对所有成交信息的断点。



之后，我们要引入2个新的概念：

存量：某一时间点的累计统计量

增量：某一时间段内，累计统计量的增加量

存量是静态的，可以理解为对累计统计量的信息进行时间切片；而增量是动态的，代表时间切片信息的变化量，所以他们二者的关系如下：

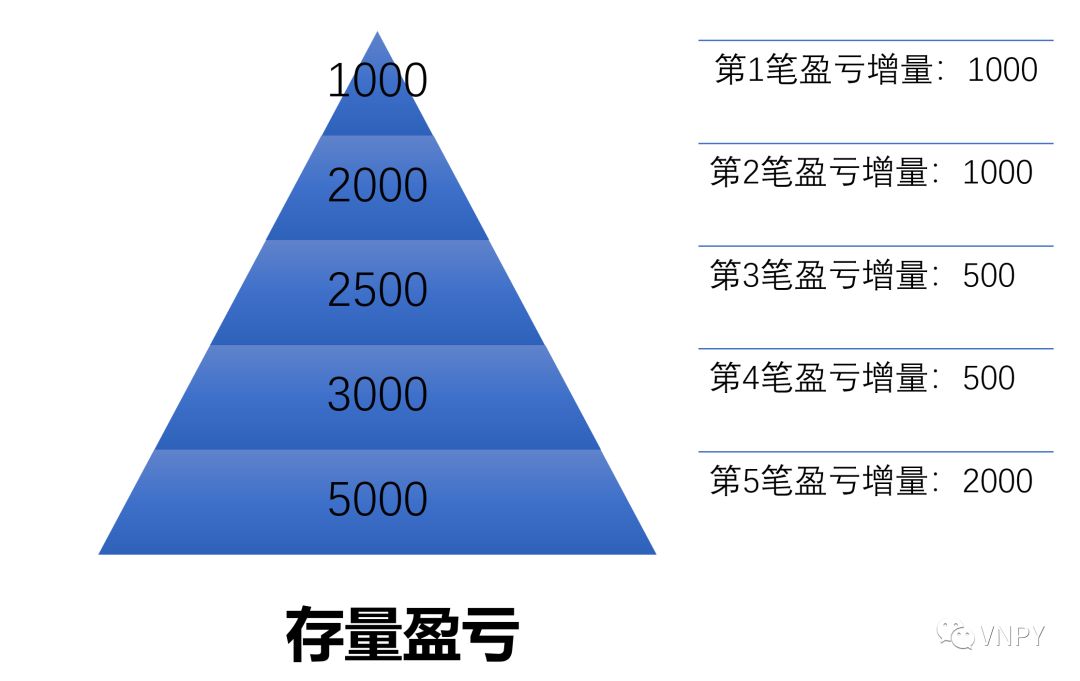
**T0时刻存量 + T0->T1增量 = T1时刻存量**

换句话说，

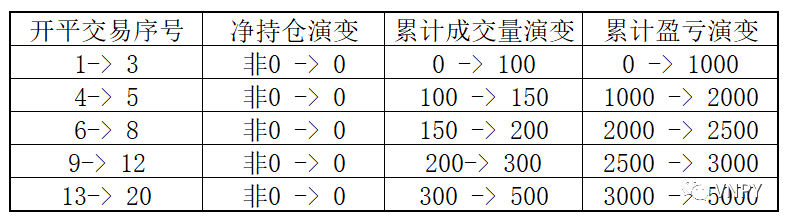
**T0->T1增量 = T1时刻存量 - T0时刻存量**

回到逐笔回测统计主题上，增量这个概念，就能代表最新的完整开平仓交易，例如其每笔盈亏，对累计盈亏的影响。

如下图所示，在完成第一笔开平仓交易后，累计盈亏是1000；完成了第二笔完整的开平仓交易，累计盈亏是2000，那么二者的差别，即2000-1000=1000。这增加1000的盈利，就是属于第二笔开平仓交易的。



所以，通过对每个断点存量信息的对比，我们就可以得到每笔开平仓成交后的统计量：



这些开平仓的统计量可以如下表所示的开平成交量、开平盈亏，也可以是开平仓交易的持仓时间、手续费、滑点以及净盈亏：



**从算法的原理到代码**

**计算开平交易结果**

生成基础DataFrame信息，包括每笔交易的方向，开平，价格，时间；

计算方向持仓，以及有方向持仓累加的净持仓，计算累计持仓存量（成交量的简单相加）；

计算盈亏存量，当净持仓为0时候，显示每笔开平交易对于存量盈亏的增量；

当净持仓为0时候，显示每笔开平交易的持仓时间，成交量，成交额的增量；

对DataFrame的行进行处理，剔除出那些净持仓不为0的行数，即剩下的行数都是每笔开平交易的最后一次平仓交易，通过平仓的方向可以判断该完整开平流程，如方向为空，开平为平，那么完整开平交易为多开->空平。

计算手续费，滑点以及净盈亏

返回新的DataFrame。

import pandas as pdfrom datetime import datetimeimport matplotlib.pyplot as pltimport numpy as nppd.set\_option('mode.chained\_assignment', None)  
  
def calculate\_trades\_result(trades): """ Deal with trade data """ dt, direction, offset, price, volume = [], [], [], [], [] for i in trades.values(): dt.append(i.datetime) direction.append(i.direction.value) offset.append(i.offset.value) price.append(i.price) volume.append(i.volume)  
 *# Generate DataFrame with datetime, direction, offset, price, volume* df = pd.DataFrame() df["direction"] = direction df["offset"] = offset df["price"] = price df["volume"] = volume  
 df["current\_time"] = dt df["last\_time"] = df["current\_time"].shift(1)  
 *# Calculate trade amount* df["amount"] = df["price"] \* df["volume"] df["acum\_amount"] = df["amount"].cumsum()  
 *# Calculate pos, net pos(with direction), acumluation pos(with direction)* def calculate\_pos(df): if df["direction"] == "多": result = df["volume"] else: result = - df["volume"]  
 return result df["pos"] = df.apply(calculate\_pos, axis=1)  
 df["net\_pos"] = df["pos"].cumsum() df["acum\_pos"] = df["volume"].cumsum()  
 *# Calculate trade result, acumulation result* *# ej: trade result(buy->sell) means (new price - old price) \* volume* df["result"] = -1 \* df["pos"] \* df["price"] df["acum\_result"] = df["result"].cumsum()  
 *# Filter column data when net pos comes to zero* def get\_acum\_trade\_result(df): if df["net\_pos"] == 0: return df["acum\_result"] df["acum\_trade\_result"] = df.apply(get\_acum\_trade\_result, axis=1)  
 def get\_acum\_trade\_volume(df): if df["net\_pos"] == 0: return df["acum\_pos"] df["acum\_trade\_volume"] = df.apply(get\_acum\_trade\_volume, axis=1)   
 def get\_acum\_trade\_duration(df): if df["net\_pos"] == 0: return df["current\_time"] - df["last\_time"] df["acum\_trade\_duration"] = df.apply(get\_acum\_trade\_duration, axis=1)   
 def get\_acum\_trade\_amount(df): if df["net\_pos"] == 0: return df["acum\_amount"] df["acum\_trade\_amount"] = df.apply(get\_acum\_trade\_amount, axis=1)   
 *# Select row data with net pos equil to zero*  df = df.dropna()  
 return df def generate\_trade\_df(trades, size, rate, slippage, capital): """ Calculate trade result from increment """ df = calculate\_trades\_result(trades)  
 trade\_df = pd.DataFrame() trade\_df["close\_direction"] = df["direction"] trade\_df["close\_time"] = df["current\_time"] trade\_df["close\_price"] = df["price"] trade\_df["pnl"] = df["acum\_trade\_result"] - \ df["acum\_trade\_result"].shift(1).fillna(0) trade\_df["volume"] = df["acum\_trade\_volume"] - \ df["acum\_trade\_volume"].shift(1).fillna(0) trade\_df["duration"] = df["current\_time"] - \ df["last\_time"] trade\_df["turnover"] = df["acum\_trade\_amount"] - \ df["acum\_trade\_amount"].shift(1).fillna(0) trade\_df["commission"] = trade\_df["turnover"] \* rate trade\_df["slipping"] = trade\_df["volume"] \* size \* slippage  
 trade\_df["net\_pnl"] = trade\_df["pnl"] - \ trade\_df["commission"] - trade\_df["slipping"]  
 result = calculate\_base\_net\_pnl(trade\_df, capital) return result

**汇总生成资金曲线**

基于每笔开平交易的净盈亏，计算累计盈亏；

累计盈亏加上用户输入的起始资金即为资金曲线；

基于资金曲线计算每笔的每笔开平交易的盈利率，回撤和百分比回撤。

def calculate\_base\_net\_pnl(df, capital): """ Calculate statistic base on net pnl """ df["acum\_pnl"] = df["net\_pnl"].cumsum() df["balance"] = df["acum\_pnl"] + capital df["return"] = np.log( df["balance"] / df["balance"].shift(1) ).fillna(0) df["highlevel"] = ( df["balance"].rolling( min\_periods=1, window=len(df), center=False).max() ) df["drawdown"] = df["balance"] - df["highlevel"] df["ddpercent"] = df["drawdown"] / df["highlevel"] \* 100  
 df.reset\_index(drop=True, inplace=True)     return df

**统计整体策略效果**

主要是一些统计指标的计算，如平均滑点，平均手续费，总成交次数，胜率，盈亏比，收益回撤比等等。

然后是画图，画出资金曲线图，每笔净盈亏图和净盈亏分布图

def statistics\_trade\_result(df, capital, show\_chart=True): """""" end\_balance = df["balance"].iloc[-1] max\_drawdown = df["drawdown"].min() max\_ddpercent = df["ddpercent"].min()  
 pnl\_medio = df["net\_pnl"].mean() trade\_count = len(df) duration\_medio = df["duration"].mean().total\_seconds()/3600 commission\_medio = df["commission"].mean() slipping\_medio = df["slipping"].mean()  
 win = df[df["net\_pnl"] > 0] win\_amount = win["net\_pnl"].sum() win\_pnl\_medio = win["net\_pnl"].mean() win\_duration\_medio = win["duration"].mean().total\_seconds()/3600 win\_count = len(win)  
 loss = df[df["net\_pnl"] < 0] loss\_amount = loss["net\_pnl"].sum() loss\_pnl\_medio = loss["net\_pnl"].mean() loss\_duration\_medio = loss["duration"].mean().total\_seconds()/3600 loss\_count = len(loss)  
 winning\_rate = win\_count / trade\_count win\_loss\_pnl\_ratio = - win\_pnl\_medio / loss\_pnl\_medio  
 total\_return = (end\_balance / capital - 1) \* 100 return\_drawdown\_ratio = -total\_return / max\_ddpercent  
 output(f"起始资金：\t{capital:,.2f}") output(f"结束资金：\t{end\_balance:,.2f}") output(f"总收益率：\t{total\_return:,.2f}%") output(f"最大回撤: \t{max\_drawdown:,.2f}") output(f"百分比最大回撤: {max\_ddpercent:,.2f}%") output(f"收益回撤比：\t{return\_drawdown\_ratio:,.2f}")  
 output(f"总成交次数:\t{trade\_count}") output(f"盈利成交次数:\t{win\_count}") output(f"亏损成交次数:\t{loss\_count}") output(f"胜率:\t\t{winning\_rate:,.2f}") output(f"盈亏比:\t\t{win\_loss\_pnl\_ratio:,.2f}")  
 output(f"平均每笔盈亏:\t{pnl\_medio:,.2f}") output(f"平均持仓小时:\t{duration\_medio:,.2f}") output(f"平均每笔手续费:\t{commission\_medio:,.2f}") output(f"平均每笔滑点:\t{slipping\_medio:,.2f}")  
 output(f"总盈利金额:\t{win\_amount:,.2f}") output(f"盈利交易均值:\t{win\_pnl\_medio:,.2f}") output(f"盈利持仓小时:\t{win\_duration\_medio:,.2f}")  
 output(f"总亏损金额:\t{loss\_amount:,.2f}") output(f"亏损交易均值:\t{loss\_pnl\_medio:,.2f}") output(f"亏损持仓小时:\t{loss\_duration\_medio:,.2f}")  
 if not show\_chart: return  
 plt.figure(figsize=(10, 12))  
 acum\_pnl\_plot = plt.subplot(3, 1, 1) acum\_pnl\_plot.set\_title("Balance Plot") df["balance"].plot(legend=True)  
 pnl\_plot = plt.subplot(3, 1, 2) pnl\_plot.set\_title("Pnl Per Trade") df["net\_pnl"].plot(legend=True)  
 distribution\_plot = plt.subplot(3, 1, 3) distribution\_plot.set\_title("Trade Pnl Distribution") df["net\_pnl"].hist(bins=100)  
 plt.show()  
  
def output(msg): """ Output message with datetime. """ print(f"{datetime.now()}\t{msg}")

**统计纯多头和纯空头交易**

纯多头交易就是只有多开->空平的交易，而纯空头交易就是反过来。

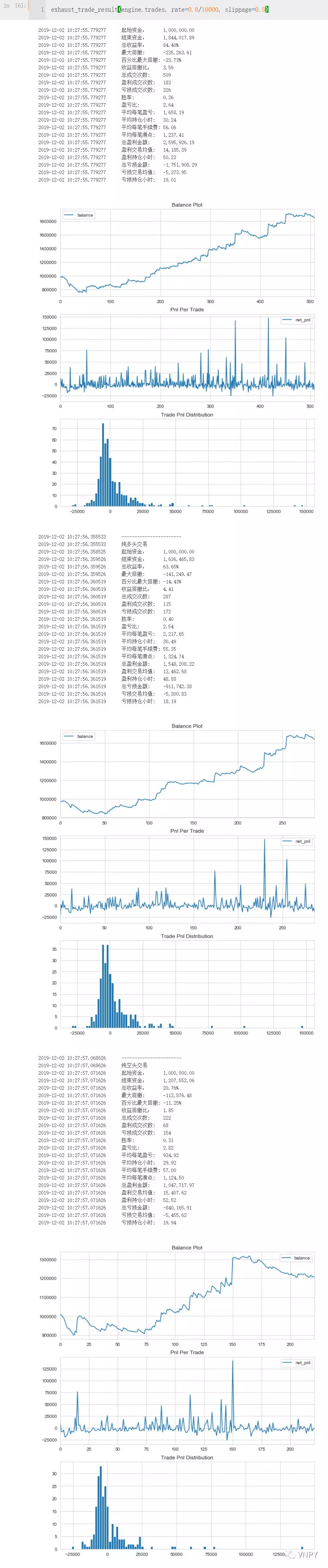
为了筛选出纯多开交易，只要在DataFrame中判断其平仓方向的空的即可；纯空头交易则反过来，平仓方向为多。

def buy2sell(df, capital): """ Generate DataFrame with only trade from buy to sell """ buy2sell = df[df["close\_direction"] == "空"] result = calculate\_base\_net\_pnl(buy2sell, capital) return result  
  
def short2cover(df, capital): """ Generate DataFrame with only trade from short to cover """ short2cover = df[df["close\_direction"] == "多"] result = calculate\_base\_net\_pnl(short2cover, capital) return result

**整合所有计算步骤**

最后，我们将上文中所有的函数进行整合，封装到单个函数中，用于实现策略回测效果的一键生成：

def exhaust\_trade\_result( trades, size: int = 10, rate: float = 0.0, slippage: float = 0.0, capital: int = 1000000, show\_long\_short\_condition=True ): """ Exhaust all trade result. """  
 total\_trades = generate\_trade\_df(trades, size, rate, slippage, capital) statistics\_trade\_result(total\_trades, capital)  
 if not show\_long\_short\_condition: return long\_trades = buy2sell(total\_trades, capital) short\_trades = short2cover(total\_trades, capital)  
 output("-----------------------") output("纯多头交易") statistics\_trade\_result(long\_trades, capital)  
 output("-----------------------") output("纯空头交易") statistics\_trade\_result(short\_trades, capital)



**最后附上完整的源代码**

import pandas as pdfrom datetime import datetimeimport matplotlib.pyplot as pltimport numpy as nppd.set\_option('mode.chained\_assignment', None)  
  
def calculate\_trades\_result(trades): """ Deal with trade data """ dt, direction, offset, price, volume = [], [], [], [], [] for i in trades.values(): dt.append(i.datetime) direction.append(i.direction.value) offset.append(i.offset.value) price.append(i.price) volume.append(i.volume)  
 *# Generate DataFrame with datetime, direction, offset, price, volume* df = pd.DataFrame() df["direction"] = direction df["offset"] = offset df["price"] = price df["volume"] = volume  
 df["current\_time"] = dt df["last\_time"] = df["current\_time"].shift(1)  
 *# Calculate trade amount* df["amount"] = df["price"] \* df["volume"] df["acum\_amount"] = df["amount"].cumsum()  
 *# Calculate pos, net pos(with direction), acumluation pos(with direction)* def calculate\_pos(df): if df["direction"] == "多": result = df["volume"] else: result = - df["volume"]  
 return result df["pos"] = df.apply(calculate\_pos, axis=1)  
 df["net\_pos"] = df["pos"].cumsum() df["acum\_pos"] = df["volume"].cumsum()  
 *# Calculate trade result, acumulation result* *# ej: trade result(buy->sell) means (new price - old price) \* volume* df["result"] = -1 \* df["pos"] \* df["price"] df["acum\_result"] = df["result"].cumsum()  
 *# Filter column data when net pos comes to zero* def get\_acum\_trade\_result(df): if df["net\_pos"] == 0: return df["acum\_result"] df["acum\_trade\_result"] = df.apply(get\_acum\_trade\_result, axis=1)  
 def get\_acum\_trade\_volume(df): if df["net\_pos"] == 0: return df["acum\_pos"] df["acum\_trade\_volume"] = df.apply(get\_acum\_trade\_volume, axis=1)   
 def get\_acum\_trade\_duration(df): if df["net\_pos"] == 0: return df["current\_time"] - df["last\_time"] df["acum\_trade\_duration"] = df.apply(get\_acum\_trade\_duration, axis=1)   
 def get\_acum\_trade\_amount(df): if df["net\_pos"] == 0: return df["acum\_amount"] df["acum\_trade\_amount"] = df.apply(get\_acum\_trade\_amount, axis=1)   
 *# Select row data with net pos equil to zero*  df = df.dropna()  
 return df  
  
def generate\_trade\_df(trades, size, rate, slippage, capital): """ Calculate trade result from increment """ df = calculate\_trades\_result(trades)  
 trade\_df = pd.DataFrame() trade\_df["close\_direction"] = df["direction"] trade\_df["close\_time"] = df["current\_time"] trade\_df["close\_price"] = df["price"] trade\_df["pnl"] = df["acum\_trade\_result"] - \ df["acum\_trade\_result"].shift(1).fillna(0) trade\_df["volume"] = df["acum\_trade\_volume"] - \ df["acum\_trade\_volume"].shift(1).fillna(0) trade\_df["duration"] = df["current\_time"] - \ df["last\_time"] trade\_df["turnover"] = df["acum\_trade\_amount"] - \ df["acum\_trade\_amount"].shift(1).fillna(0) trade\_df["commission"] = trade\_df["turnover"] \* rate trade\_df["slipping"] = trade\_df["volume"] \* size \* slippage  
 trade\_df["net\_pnl"] = trade\_df["pnl"] - \ trade\_df["commission"] - trade\_df["slipping"]  
 result = calculate\_base\_net\_pnl(trade\_df, capital) return result  
  
def calculate\_base\_net\_pnl(df, capital): """ Calculate statistic base on net pnl """ df["acum\_pnl"] = df["net\_pnl"].cumsum() df["balance"] = df["acum\_pnl"] + capital df["return"] = np.log( df["balance"] / df["balance"].shift(1) ).fillna(0) df["highlevel"] = ( df["balance"].rolling( min\_periods=1, window=len(df), center=False).max() ) df["drawdown"] = df["balance"] - df["highlevel"] df["ddpercent"] = df["drawdown"] / df["highlevel"] \* 100  
 df.reset\_index(drop=True, inplace=True) return df  
  
def buy2sell(df, capital): """ Generate DataFrame with only trade from buy to sell """ buy2sell = df[df["close\_direction"] == "空"] result = calculate\_base\_net\_pnl(buy2sell, capital) return result  
  
def short2cover(df, capital): """ Generate DataFrame with only trade from short to cover """ short2cover = df[df["close\_direction"] == "多"] result = calculate\_base\_net\_pnl(short2cover, capital) return result  
  
def statistics\_trade\_result(df, capital, show\_chart=True): """""" end\_balance = df["balance"].iloc[-1] max\_drawdown = df["drawdown"].min() max\_ddpercent = df["ddpercent"].min()  
 pnl\_medio = df["net\_pnl"].mean() trade\_count = len(df) duration\_medio = df["duration"].mean().total\_seconds()/3600 commission\_medio = df["commission"].mean() slipping\_medio = df["slipping"].mean()  
 win = df[df["net\_pnl"] > 0] win\_amount = win["net\_pnl"].sum() win\_pnl\_medio = win["net\_pnl"].mean() win\_duration\_medio = win["duration"].mean().total\_seconds()/3600 win\_count = len(win)  
 loss = df[df["net\_pnl"] < 0] loss\_amount = loss["net\_pnl"].sum() loss\_pnl\_medio = loss["net\_pnl"].mean() loss\_duration\_medio = loss["duration"].mean().total\_seconds()/3600 loss\_count = len(loss)  
 winning\_rate = win\_count / trade\_count win\_loss\_pnl\_ratio = - win\_pnl\_medio / loss\_pnl\_medio  
 total\_return = (end\_balance / capital - 1) \* 100 return\_drawdown\_ratio = -total\_return / max\_ddpercent  
 output(f"起始资金：\t{capital:,.2f}") output(f"结束资金：\t{end\_balance:,.2f}") output(f"总收益率：\t{total\_return:,.2f}%") output(f"最大回撤: \t{max\_drawdown:,.2f}") output(f"百分比最大回撤: {max\_ddpercent:,.2f}%") output(f"收益回撤比：\t{return\_drawdown\_ratio:,.2f}")  
 output(f"总成交次数:\t{trade\_count}") output(f"盈利成交次数:\t{win\_count}") output(f"亏损成交次数:\t{loss\_count}") output(f"胜率:\t\t{winning\_rate:,.2f}") output(f"盈亏比:\t\t{win\_loss\_pnl\_ratio:,.2f}")  
 output(f"平均每笔盈亏:\t{pnl\_medio:,.2f}") output(f"平均持仓小时:\t{duration\_medio:,.2f}") output(f"平均每笔手续费:\t{commission\_medio:,.2f}") output(f"平均每笔滑点:\t{slipping\_medio:,.2f}")  
 output(f"总盈利金额:\t{win\_amount:,.2f}") output(f"盈利交易均值:\t{win\_pnl\_medio:,.2f}") output(f"盈利持仓小时:\t{win\_duration\_medio:,.2f}")  
 output(f"总亏损金额:\t{loss\_amount:,.2f}") output(f"亏损交易均值:\t{loss\_pnl\_medio:,.2f}") output(f"亏损持仓小时:\t{loss\_duration\_medio:,.2f}")  
 if not show\_chart: return  
 plt.figure(figsize=(10, 12))  
 acum\_pnl\_plot = plt.subplot(3, 1, 1) acum\_pnl\_plot.set\_title("Balance Plot") df["balance"].plot(legend=True)  
 pnl\_plot = plt.subplot(3, 1, 2) pnl\_plot.set\_title("Pnl Per Trade") df["net\_pnl"].plot(legend=True)  
 distribution\_plot = plt.subplot(3, 1, 3) distribution\_plot.set\_title("Trade Pnl Distribution") df["net\_pnl"].hist(bins=100)  
 plt.show()  
  
def output(msg): """ Output message with datetime. """ print(f"{datetime.now()}\t{msg}")  
  
def exhaust\_trade\_result( trades, size: int = 10, rate: float = 0.0, slippage: float = 0.0, capital: int = 1000000, show\_long\_short\_condition=True ): """ Exhaust all trade result. """  
 total\_trades = generate\_trade\_df(trades, size, rate, slippage, capital) statistics\_trade\_result(total\_trades, capital)  
 if not show\_long\_short\_condition: return long\_trades = buy2sell(total\_trades, capital) short\_trades = short2cover(total\_trades, capital)  
 output("-----------------------") output("纯多头交易") statistics\_trade\_result(long\_trades, capital)  
 output("-----------------------") output("纯空头交易") statistics\_trade\_result(short\_trades, capital)