There are two ways to get the arb data -

1. With volume consideration while calculating arbitrage -
   This data is fetched when with_volume is true while subscribing to arbitrage data via websocket
2. Without any volume consideration in calculating arbitrage -
   This data is fetched when with_volume is false while subscribing to arbitrage data via websocket

When with_volume is false, below are the fields to be considered while doing arbitrage -

- BE - Buy Exchange Name from where coin needs to be bought
- BEUC - Buy Exchange unique code from where coin needs to be bought
- SE - Sell Exchange from where coin needs to be sold
- SEUC - Sell Exchange unique code from where coin needs to be sold
- BC - Coin which needs to be bought
- BCUC - Coin unique code which needs to be bought
- AC - Against currency in which price is calculated.
- BCC - Volume to buy. This volume is assumed one and is not checked if such volume is really present or not.
- SCC - Volume to sell. This volume is assumed one and is not checked if such volume is really present or not.
- BM - Market in which coin needs to be bought
- SM - Market in which coin needs to be sold
- CH - Detailed charges to be incurred in a arbitrage
- BTC - Buy trading charges
- BTCP - Buy trading percentage
- STC - Sell trading charges
- STCP - Sell trading percentage
- TRC - Transfer charges.
- TRCC - Transfer cost which is usually in fraction of the buy currency
- MBPU - Buy price per unit in BM. **This is the price where you need to buy the desired volume**
- MSPU - Sell price per unit in SM. **This is the price where you need to sell the desired volume**
- BPIC - Buy price inclusive of all charges
- BP - Buy price exclusive of all charges
- SPIC - Sell price inclusive of all charges
- SP - Sell price exclusive of all charges
- A - Total Arbitrage profit
- AP - Arbitrage percentage
- C - Unix timestamp when arbitrage was created

When with_volume is true, the arbitrage considers available volume in orderbook. Below are the fields to be considered while doing arbitrage -

- BE - Buy Exchange Name from where coin needs to be bought
- BEUC - Buy Exchange unique code from where coin needs to be bought
- SE - Sell Exchange from where coin needs to be sold
- SEUC - Sell Exchange unique code from where coin needs to be sold
- BC - Coin which needs to be bought
- BCUC - Coin unique code which needs to be bought
- AC - Against currency in which price is calculated.
- TVTB - Volume to buy.
- TVTS - Volume to sell.
- BM - Market in which coin needs to be bought
- SM - Market in which coin needs to be sold
- CH - Trading percentage and transfer cost to be incurred in a arbitrage
- CHV - Detailed charges in to be incurred in a arbitrage
- BTCV - Buy trading charges.
- STCV - Sell trading charges.
- TRCV - Transfer charges. This is usually TRCC * SPU
- BTCP - Buy trading percentage
- STCP - Sell trading percentage
- TRCC - Transfer cost which is usually in fraction of the buy currency
- MPMBPU - Minimum profitable market buy price per unit. **This is the price where you need to buy the desired volume**
- MPMSPU - Minimum profitable market sell price per unit. **This is the price where you need to sell the desired volume**
- BSOB - Buy Side order book. It has array of buy orders needed to be executed for completing the arb. Structure is in the form of [[p1,v1],[p2,v2]....] where p is price and v is volume.
- SSOB - Sell Side order book. It has array of sell orders needed to be executed for completing the arb. Structure is in the form of [[p1,v1],[p2,v2]....] where p is price and v is volume.
- CBPIC - Cumulative Buy price inclusive of all charges
- CBP - Cumulative Buy price exclusive of all charges
- CSPIC - Cumulative Sell price inclusive of all charges
- CSP - Cumulative Sell price exclusive of all charges
- MP - Total Arbitrage profit
- MPP - Arbitrage percentage
- C - Unix timestamp when arbitrage was created

Following is an example of Direct/Intra-exchange Arbitrage


```
{
  BE: 'Kucoin',
  SE: 'Binance',
  BC: 'WTC',
  AC: 'BTC',
  BCC: 80,
  SCC: 79.5,
```

BM: 'BTC',
SM: 'BTC',
CH:
 {
   BTC: 0.0000252792,
   BTCP: 0.1,
   STC: 0.0000269981999999995,
   STCP: 0.1,
   TRC: 0.0001698,
   TRCC: 0.5
 },
BPU: 0.00031599,
BMLP: 0.00031546,
SMLP: 0.0003397,
MBPU: 0.00031599,
BPIC: 0.025304479199999997,
BP: 0.025279199999999998,
MSPU: 0.0003396,
SPU: 0.0003396,
SPIC: 0.0269712018,
SP: 0.027168,
SSR: 108.46687572999994,
BSR: 0,
TVTB: 488.60312427,
TVTS: 488.10312427,
CBP: 0.1617144475591,
CBPIC: 0.1618761620066591,
CSP: 0.165694703439957,
CSPIC: 0.16535962828651704,
BI: 5,
SI: 3,
MP: 0.003483296729857952,
A: 0.0016667226000000035,
AP: '6.59',
CHV:
 { TRCV: 0.00016955,
   BTCV: 0.0001617144475591,
   STCV: 0.000165525153439957 },
BEM: 1,
SEM: 1,
MPP: 2.151828092955811,
C: 1554491030586,
MPMBPU: 0.00033491,
MPMSPU: 0.0003391,
BSOB: [[0.00031599, 100], [0.00031598, 90]...],
SSOB: [[0.0003396, 50], [0.0003397, 80]...]

}

Triangular/loop arbitrage consists of 2 step arbitrages. When with_volume flag is false, following fields to be considered

- AC - Against currency in which price is calculated.
- AP - Arbitrage percentage
- A - Arbitrage profit
- ICC - Intermediate coin count fetched when user sells coin in exchange and buys another coin with the intermediate coin count. For example - If the sell market of ARB1 is BTC, then intermediate coin will be in BTC. Using that BTC, user can buy another coin
- C - Unix timestamp when arbitrage was created
- ARB1 - As triangular arbitrage consists of two direct arbitrages, it is the first step arbitrage. **It has the same structure as direct arbitrage**
- ARB2 - As triangular arbitrage consists of two direct arbitrages, it is the second step arbitrage. **It has the same structure as direct arbitrage**

When with_volume flag is true, following fields to be considered -

- AC - Against currency in which price is calculated.
- MPP - Arbitrage percentage
- MP - Arbitrage profit
- ICCWV - Intermediate coin count fetched when user sells coin in exchange and buys another coin with the intermediate coin count. For example - If the sell market of ARB1 is BTC, then intermediate coin will be in BTC. Using that BTC, user can buy another coin
- C - Unix timestamp when arbitrage was created
- ARB1 - As triangular arbitrage consists of two direct arbitrages, it is the first step arbitrage. It has the same structure as direct arbitrage
- ARB2 - As triangular arbitrage consists of two direct arbitrages, it is the second step arbitrage. It has the same structure as direct arbitrage

Following is an example of triangular/loop arbitrage -

{
  AC: 'BTC',
  MPP: 1.8316682402140816,
  MP: 0.0029650342479534686,
  AP: 6.258938592218439,
  A: 0.0015837918142086876,
  ARB1:
   { BE: 'Kucoin',
     SE: 'Binance',
     BC: 'WTC',
     AC: 'BTC',
     BCC: 80,

```
  SCC: 79.5,
  BM: 'BTC',
  SM: 'BTC',
  CH:
      { BTC: 0.0000252792,
        BTCP: 0.1,
        STC: 0.00004054499999999947,
        STCP: 0.15,
        TRC: 0.00017,
        TRCC: 0.5 },
  BPU: 0.00031599,
  BMLP: 0.00031546,
  SMLP: 0.0003397,
  MBPU: 0.00031599,
  BPIC: 0.025304479199999997,
  BP: 0.025279199999999998,
  MSPU: 0.0003396,
  SPU: 0.0003396,
  SPIC: 0.0269712018,
  SP: 0.027168,
  SSR: 108.46687572999996,
  BSR: 0,
  TVTB: 488.60312426999997,
  TVTS: 488.10312426999997,
  CBP: 0.1617144475591,
  CBPIC: 0.1618761620066591,
  CSP: 0.165694703439957,
  CSPIC: 0.16535962828651704,
  BI: 5,
  SI: 3,
  MP: 0.003483466279857933,
  A: 0.0016667226000000035,
  AP: '6.59',
  CHV:
      { TRCV: 0.0001685,
        BTCV: 0.00003191318275947055,
        STCV: 0.000048739111230189896 },
  BEM: 1,
  SEM: 1,
  MPP: 2.151932833516669,
  C: 1554555961020,
  MPMBPU: 0.00033491,
  MPMSPU: 0.0003391,
  BSOB: [[0.00031599, 100], [0.00031598, 90]...],
  SSOB: [[0.0003396, 50], [0.0003397, 80]...],
},
```

```
ARB2:
 { BE: 'Binance',
   SE: 'OKEx',
   BC: 'XRP',
   AC: 'BTC',
   BCC: 352.3967766473652,
   SCC: 352.1467766473652,
   BM: 'BTC',
   SM: 'BTC',
   CH:
      { BTC: 0.000040423547179233715,
        BTCP: 0.15,
        STC: 0.00004810139732176061,
        STCP: 0.18,
        TRC: 2.6040000000000003e-7,
        TRCC: '0.01' },
   BPU: 0.00007646,
   BMLP: 0.00007646,
   SMLP: 0.00007647,
   MBPU: 0.00007646,
   BPIC: 0.0269712018,
   BP: 0.026944257542457547,
   MSPU: 0.00007647,
   SPU: 0.00007647,
   SPIC: 0.026888271014208685,
   SP: 0.026947781510224016,
   SSR: 1685.9993239290834,
   BSR: 8100.313323929084,
   TVTB: 2159.6866760709163,
   TVTS: 2159.4366760709163,
   CBP: 0.16519443385266439,
   CBPIC: 0.16535962828651704,
   CSP: 0.16510793949886085,
   CSPIC: 0.16484119625461258,
   BI: 0,
   SI: 3,
   MP: -0.0005184320319044644,
   A: -0.00008293078579131585,
   AP: '-0.31',
   CHV:
      { TRCV: 2.603e-7,
        BTCV: 0.000048593112894003615,
        STCV: 0.00005780123086711464 },
   BEM: 1,
   SEM: 1,
   MPP: -0.31351789870147884,
```

```
    C: 1554555961020,
    MPMBPU: 0.00007649,
    MPMSPU: 0.00007644,
    BSOB: [[0.00007646, 100], [0.00007645, 90]...],
    SSOB: [[0.00007647, 50], [0.00007648, 80]...],
  },
  ICC: 0.0269712018,
  ICCWV: 0.16535962828651704,
  C: 1554555961294
}
```

Points to note while doing arbitrage -

1. Always check arbitrage **creation unix timestamp** is always within 1 minute old. If it is more than 1 minute old, then please do not execute the arbitrage and report to the KoinKnight team
2. Always check if the arbitrage **Buy and Sell price per unit** is providing you the profit before executing it.

Below is the sample NodeJS code

```
let io = require('socket.io-client'); // version 3.0.1
var crypto = require('crypto');
var zlib = require('zlib');

let socket;
let apiKey = '<Your API Key>';
let apiSecret = '<Your API Secret>';
let socketUrl = 'https://api.koinknight.com/api';
let arbitrageType = 'direct_arbitrage'; // This value can be direct_arbitrage,
// triangular_arbitrage, loop_arbitrage and intra_exchange_arbitrage
let withVolume = true; // If this flag is true, then volume is considered in arb calculation.
let sortBy = 'profit'; // Default sort is percentage if not passed any value. It only works when
withVolume is true
let sortOrder = 'asc'; // Default order is descending if not passed any value. It only works when
withVolume is true
let market = 'BTC'; // Market can be BTC, ETH, USDT
let disabledWallets =false;

function getListenMessageKey() {
  let listenMessageKey = arbitrageType;
  if (arbitrageType === 'direct_arbitrage' && disabledWallets) {
    listenMessageKey =  `disabled_${listenMessageKey}`;
  }
  if (withVolume) {
    listenMessageKey += `_volume`;
    if (sortBy === 'profit') {
      listenMessageKey += `_${sortBy}`;
    }
  }
  listenMessageKey += `_${market}`;
```

```
  listenMessageKey += `_${apiKey}`;
  if (withVolume && sortOrder === 'asc') {
     listenMessageKey += `_${sortOrder}`;
  }
  return listenMessageKey;
}

function getSignature() {

  let timestamp = Date.now();
  var hmac = crypto.createHmac('sha256', apiSecret);
  hmac.update(timestamp + apiKey);
  let signature = hmac.digest('hex');
  return {signature: signature, timestamp: timestamp};
}

function init() {
  let signatureData = getSignature();
  let listenMessageKey = getListenMessageKey();
  socket = io(socketUrl, {
     extraHeaders: {
        'x-koinknight-apikey': apiKey,
        'x-koinknight-signature': signatureData.signature,
        'x-koinknight-timestamp': signatureData.timestamp
     }

  });


  socket.on('connect', () => {
     logger.info("Socket Connected");
     // subscribe to api for direct arbitrage. Top 200 arbitrages will be emitted
     socket.emit('kk_api_subscribe', {
        "kk_room":"arbitrages",
        "arbitrage_type": arbitrageType,
        "market": market,
        "with_volume": withVolume,
        "sort_order": sortOrder,
        "sort_by": sortBy
     });

     // If you want to unsubscribe to the channel, then pass the event name kk_api_unsubscribe with
the same data
     // socket.emit('kk_api_unsubscribe', {
     //     "kk_room":"arbitrages",
     //     "arbitrage_type": arbitrageType,
     //     "market": market,
     //     "with_volume": withVolume,
     //     "sort_order": sortOrder,
     //     "sort_by": sortBy
     // });
```

```
  });


  socket.on('disconnect', () => {
    logger.info("Socket Disconnected");
    socket.io.reconnect();
  });

  socket.on(listenMessageKey, (msg) => {
    let data = JSON.parse(zlib.inflateSync(new Buffer(msg.data, 'base64')).toString());
    console.log(data)
  })

  socket.on('error', (err) => {
    logger.error("Error in socket :: %s", err);
  })

  socket.on('custom_error', (err) => {
    logger.error("Error in socket :: %s", err);
  })

}

export default function () {
  init();
}
```

Below is the sample Python Code(version 3.6.8)

```
import socketio # Install python-socketio[client]===5.x
import time
from Crypto.Hash import HMAC, SHA256 # Install pycrypto
import json
import zlib
import base64

apiKey = 'e4c8611509ed7ac5dafafb6ce0fa3da8'
apiSecret = '4676f14b3e587c8d9a162f35f6c0a6866e310c48cdec2ae078e90ec4d206854e'
socketUrl = 'https://api.koinknight.com'
arbitrageType = 'direct_arbitrage' # This value can be direct_arbitrage,
# triangular_arbitrage, loop_arbitrage, intra_exchange_arbitrage and intra_exchange_loop_arbitrage
withVolume = True # If this flag is true, then volume is considered in arb calculation. For
intra_exchange_loop_arbitrage, always keep true for this field.
sortBy = 'profit' # Default sort is percentage if not passed any value. It only works when withVolume is
true
sortOrder = 'asc' # Default order is descending if not passed any value. It only works when
withVolume is true
market = 'INR' # Market can be BTC, ETH, USDT
disabledWallets = False
```

```
def getListenMessageKey():
  listenMessageKey = arbitrageType
  if arbitrageType == 'direct_arbitrage' and disabledWallets:
     listenMessageKey =  'disabled_' + listenMessageKey
  if withVolume:
    listenMessageKey += '_volume'
    if sortBy == 'profit':
       listenMessageKey += '_%s' %sortBy

  listenMessageKey += '_%s' %market
  listenMessageKey += '_%s' %apiKey
  if withVolume and sortOrder == 'asc':
    listenMessageKey += '_%s' %sortOrder
  return listenMessageKey


def getSignature():
  timestamp = int(time.time() * 1000)
  msg = '%s%s' % (timestamp, apiKey)
  hmac = HMAC.new(apiSecret.encode('utf-8'), digestmod=SHA256)
  hmac.update(msg.encode('utf-8'))
  signature = hmac.hexdigest()
  return {'signature': signature, 'timestamp': str(timestamp)}

def init():
  signatureData = getSignature()
  listenMessageKey = getListenMessageKey()
  sio = socketio.Client()

  @sio.on('connect', namespace='/api')
  def connect_handler():
     print('Connected!')
     sio.emit('kk_api_subscribe', {
        "kk_room": "arbitrages",
        "arbitrage_type": arbitrageType,
        "market": market,
        "with_volume": withVolume,
        "sort_order": sortOrder,
        "sort_by": sortBy
     }, namespace='/api')

  @sio.on(listenMessageKey, namespace='/api')
  def on_message_handler(msg):
     print(zlib.decompress(base64.b64decode(msg['data'])))

  @sio.on('disconnect', namespace='/api')
  def disconnect_handler():
     print('Disconnected!')
     sio.emit('kk_api_unsubscribe', {
        "kk_room": "arbitrages",
        "arbitrage_type": arbitrageType,
```

```
            "market": market,
            "with_volume": withVolume,
            "sort_order": sortOrder,
            "sort_by": sortBy
        }, namespace='/api')

    @sio.on('error', namespace='/api')
    def error_handler(err):
        print(err)

    @sio.on('custom_error', namespace='/api')
    def custom_error_handler(err):
        print(err)


    sio.connect(socketUrl, namespaces=['/api'], headers={
      'x-koinknight-apikey': apiKey,
      'x-koinknight-signature': signatureData['signature'],
      'x-koinknight-timestamp': signatureData['timestamp']
    })
    sio.wait()


if __name__ == '__main__':
    init()
```