Python API tutorial

1) Introduction

Short tutorial on how to access the Winton Climate Prediction Market via the API using python. We tested this on python 3.6.1

Required python packages

1. requests
2. json

Your personal API Token for your account is available at https://app.climatepredictionmarket.com/account (https://app.climatepredictionmarket.com/account).

The API can be seen at https://api.climatepredictionmarket.com/swagger (https://api.climatepredictionmarket.com/swagger) (Authenticate using 'Bearer [*API Token*]' to see authorized endpoints).

1.1) Helper functions

We begin by defining a helper class that wraps up the all the basic functionality we will need.

In [34]:

```
import requests as _requests
import json as _json

#from IPython.core.display import display, HTML
#display(HTML("<style>.container { width:100% !important; }</style>"))

class Client(object):

    def __init__(self, baseurl, token, verbose=False):
        self._baseurl = baseurl
        self._verbose = verbose
        self._token = token

    def __enter__(self):
        self._session = _requests.Session()
        #self._session.verify = 'location to SSL certificates if you need them'
        self._session.headers.update({'Content-Type': 'application/json-patch+json'})
        self._session.headers.update({'Accept': 'application/json', 'Authorization': 'B
earer {}'.format(self._token)})
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        self._session.close()

    def get(self, url, params = ''):
        resource = '{}/{}/{}'.format(self._baseurl, url, params)
        if self._verbose:
            print('GET {}\n'.format(resource))
        return self._handle(self._session.get(resource))

    def post(self, url, data={}):
        resource = '{}/{}'.format(self._baseurl, url)
        payload = _json.dumps(data)
        if self._verbose:
            print('POST {}\n{}\n'.format(resource, payload))
        return self._handle(self._session.post(resource, payload))

    def _handle(self, response):
        if response.status_code/100 == 2:
            if response.content:
                if self._verbose:
                    print('{}\n'.format(response.content))
                return _json.loads(response.content)
        else:
            try:
                message = _json.loads(response.content)['message']
            except:
                message = response.reason
            raise IOError(message)
```

In [ ]:

```
baseurl = "https://api.climatepredictionmarket.com"
token = "ey0123456789abcdef..." # replace this with your API Token
```

2) Get ids

Everything in the market is indexed by various ids. The most important ones are :

1. **marketId** : identifies a particular market (e.g. June 2018).
2. **eventSpaceId** : for a given market, identifies the entire available outcome space. There is a one-to-one mapping between marketId and eventSpaceId.
3. **eventId** : for a given outcome space, identifies a particular event (e.g. the cell with `0 <= temperature < 0.2 and 0 <= rainfall < 10`, in the June 2018 market, will have a unique eventId).
4. **variableId** : identifies the market variables (Temperature/ Rainfall).
5. **accountId** : identifies your account.
6. **contractId** : identifies any contracts that you create.

In [36]:

```
#start a session, read various ids

with Client(baseurl, token, False) as agent:
    markets = agent.get('Market')
    variables = agent.get('Variable')
    market = next(m for m in markets if m['expiry'] == '2018-04')
    variableIdT = next(v for v in variables if v['code'] == 'UK_TEMP')['id']
    variableIdR = next(v for v in variables if v['code'] == 'UK_RAIN')['id']
```

In [37]:

```
markets
```

Out[37]:


```
[{'eventSpaceId': '54ff1cfe-ad53-4f94-a4fe-dd82588f75c0',
  'expiry': '2018-04',
  'id': '72a6c511-41ba-462f-ae66-148b477259e6',
  'liquidityFactor': 500.0,
  'settled': False,
  'spreadCoefficients': {'constant': 0.001, 'factor': 0.001}},
 {'eventSpaceId': '54ff1cfe-ad53-4f94-a4fe-dd82588f75c0',
  'expiry': '2018-05',
  'id': '76770eea-e751-46f9-8ec5-20f39a54a428',
  'liquidityFactor': 500.0,
  'settled': False,
  'spreadCoefficients': {'constant': 0.001, 'factor': 0.001}},
 {'eventSpaceId': '54ff1cfe-ad53-4f94-a4fe-dd82588f75c0',
  'expiry': '2018-06',
  'id': '7e737048-0748-41de-acd4-171505f9165a',
  'liquidityFactor': 500.0,
  'settled': False,
  'spreadCoefficients': {'constant': 0.001, 'factor': 0.001}},
 {'eventSpaceId': '54ff1cfe-ad53-4f94-a4fe-dd82588f75c0',
  'expiry': '2018-07',
  'id': '4266b8dd-a65d-4805-ad6a-60c2a0b319f9',
  'liquidityFactor': 500.0,
  'settled': False,
  'spreadCoefficients': {'constant': 0.001, 'factor': 0.001}},
 {'eventSpaceId': '54ff1cfe-ad53-4f94-a4fe-dd82588f75c0',
  'expiry': '2018-08',
  'id': 'fd1c13d3-6904-484c-912c-700a5f5142da',
  'liquidityFactor': 500.0,
  'settled': False,
  'spreadCoefficients': {'constant': 0.001, 'factor': 0.001}},
 {'eventSpaceId': '54ff1cfe-ad53-4f94-a4fe-dd82588f75c0',
  'expiry': '2018-09',
  'id': '29156ee1-30ff-4b58-831a-2c96f5e3c01b',
  'liquidityFactor': 500.0,
  'settled': False,
  'spreadCoefficients': {'constant': 0.001, 'factor': 0.001}}]
```


3) Check account

The **Account** endpoint includes details such as balance, contracts in your portfolio etc.

In [38]:

```
account = agent.get('Account') # check account details
account
```

Out[38]:


```
{'balance': 1000.0,
 'deposited': 1000.0,
 'id': '3fdcca65-cd82-462e-afd0-d7901a1d34f8',
 'lastTransactionId': 1,
 'portfolio': [],
 'withdrawn': 0.0}
```

4) Get market price distribution and marginal distributions for the market variables

1. The **Price** endpoint requires a marketId and returns the price for each cell indexed by the cell eventId
2. The **ProbabilityDistribution** endpoint requires a variableId (such as the one for temperature) and an expiry date (like 2018-05, note that there is a one-to-one mapping between expiry date and marketId in our market) and returns the marginal distribution for the variable in the given market.

In [39]:

```
prices = agent.get('Price', market['id']) # full 2-d price distribution

dates = agent.get('StatsMetadata/Dates', '{}'.format(variableIdT)) # get expiry dates f
or a given variableId
TempDistribution = agent.get('ProbabilityDistribution', '{}/{}'.format(variableIdT, dat
es[0]))
```

5) Market Order Status

The **MarketOrderStatus** will tell you which markets are open and what type of orders are they accepting. The output is indexed by marketId

In [40]:

```
agent.get('MarketOrderStatus')
```

Out[40]:

```
[{'isOpenForAuctionOrders': False,
  'isOpenForRealTimeOrders': True,
  'isProcessingAuction': False,
  'marketExpiry': '2018-04',
  'marketId': '72a6c511-41ba-462f-ae66-148b477259e6'},
 {'isOpenForAuctionOrders': False,
  'isOpenForRealTimeOrders': False,
  'isProcessingAuction': False,
  'marketExpiry': '2018-05',
  'marketId': '76770eea-e751-46f9-8ec5-20f39a54a428'},
 {'isOpenForAuctionOrders': False,
  'isOpenForRealTimeOrders': False,
  'isProcessingAuction': False,
  'marketExpiry': '2018-06',
  'marketId': '7e737048-0748-41de-acd4-171505f9165a'},
 {'isOpenForAuctionOrders': False,
  'isOpenForRealTimeOrders': False,
  'isProcessingAuction': False,
  'marketExpiry': '2018-07',
  'marketId': '4266b8dd-a65d-4805-ad6a-60c2a0b319f9'},
 {'isOpenForAuctionOrders': False,
  'isOpenForRealTimeOrders': False,
  'isProcessingAuction': False,
  'marketExpiry': '2018-08',
  'marketId': 'fd1c13d3-6904-484c-912c-700a5f5142da'},
 {'isOpenForAuctionOrders': False,
  'isOpenForRealTimeOrders': False,
  'isProcessingAuction': False,
  'marketExpiry': '2018-09',
  'marketId': '29156ee1-30ff-4b58-831a-2c96f5e3c01b'}]
```

6) Create a contract

To create a contract you need the following :

1. **marketId** for the market you want to create the contract in.
2. **contract name** : string identifier.
3. **list of weights** : where each entry in the list should have an **eventId** for some cell in the outcome space and a **value** which is the relavtive weight assigend to that outcome by you.

You will also need to make sure that the market is open and accepting orders (for creating contracts, either **isOpenForRealTimeOrders** or **isOpenForAuctionOrders** in the above should be **True**)

In [41]:

```
#for a given marketId get the the full event space.
eventSpace = agent.get('Event/EventSpace', market['eventSpaceId'])
```

In [42]:

```
#helper function to extract the eventId for a given value of temperatue and rainfall
def isEventFor(event, temperature, rainfall):
            intervalT = next(i for i in event['intervals'] if i['variableId'] == vari
ableIdT)
            intervalR = next(i for i in event['intervals'] if i['variableId'] == vari
ableIdR)

            if intervalT['low'] == None:
                intervalT['low'] = -float('inf')

            if intervalR['low'] == None:
                intervalR['low'] = -float('inf')

            if intervalT['high'] == None:
                intervalT['high'] = float('inf')

            if intervalR['high'] == None:
                intervalR['high'] = float('inf')

            return intervalT['low'] <= temperature < intervalT['high'] and intervalR[
'low'] <= rainfall < intervalR['high']
```

In [43]:

```
#get the eventIds for a few events that we will include in our contract
e1 = next(e['id'] for e in eventSpace if isEventFor(e, -10.0, 750.0))
e2 = next(e['id'] for e in eventSpace if isEventFor(e, 10.0, 500.0))
e3 = next(e['id'] for e in eventSpace if isEventFor(e, 20.0, 50.0))
e4 = next(e['id'] for e in eventSpace if isEventFor(e, 0.1, 110.0))
```

In [44]:

```
#create a contract by posting to the Contract endpoint. The call returns a contract id
 uniquely identifying your contract.
c1 = agent.post('Contract',
    {
        'name': 'Sample contract',
        'marketId': market['id'],
        'weights': [
            {'eventId': e1, 'value': 0.5},
            {'eventId': e2, 'value': 1.0},
            {'eventId': e3, 'value': 0.2},
            {'eventId': e4, 'value': 1.2}
        ]
    })
```

In [45]:

```
c1
```

Out[45]:

```
'97e30765-2b7c-40bb-a53c-2ac60ad10561'
```

7) Get a buy quote for a real time order

For this we will use the **Quote** endpoint. This requires :

1. **contractId** : which you can either get from the previous step if you are creating a new contract of or check the **Account** endpoint for a list of all contractIds. You can use the **GET /Contract/{id}** endpoint to get all details of your contracts including name, weights etc.
2. **quantity** : > 0 for buying, < 0 for selling (short selling not allowed)

In [46]:

```
price = agent.post('Quote',
          [
              {
                  "contractId":c1,
                  'quantity' : 10

              }
          ]

          )
price
```

Out[46]:

```
0.006613109758936844
```

8) Post Real time order

We use the **RealTimeOrder** endpoint which will require contractId, quantity, and slippage limits. Real time orders should be settled instantaneously, but you can query the order status using the **GET /RealTimeOrder/Market/{marketId}/Active** and **GET /RealTimeOrder/Market/{marketId}/Complete** endpoints.

In [47]:

```
agent.post('RealTimeOrder',
            {
              "contractQuantities": [
                {
                  "contractId": c1,
                  "quantity": 10
                }
              ],
              "limit": 0.01
            }
          )
```

Out[47]:

```
'e8dc99e8-1f52-44cf-9d67-0b8f765d5e89'
```

In [48]:

```
#view updated account
account = agent.get('Account')
account
```

Out[48]:

```
{'balance': 999.993386890241,
 'deposited': 1000.0,
 'id': '3fdcca65-cd82-462e-afd0-d7901a1d34f8',
 'lastTransactionId': 2,
 'portfolio': [{'contractId': '97e30765-2b7c-40bb-a53c-2ac60ad10561',
   'quantity': 10}],
 'withdrawn': 0.0}
```

In [ ]:

```

```