|  |  |
| --- | --- |
|  | **Artificial Intelligence Logbook**  **BACHELOR'S PROJECT** |

|  |  |
| --- | --- |
| **Name student** | **Student number** |
| **Sameer Baruwal** | **20203834** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Application(s) used** | **Prompts used (and customizations/iterations)** | **Screenshot output** |
| March | ChatGPT | what is pandas in python |  |
| March | ChatGPT | what is a market clearing? |  |
| March | ChatGPT | i have put the pandas series with each hour and the load in the agentDemand and added the demand in the forecast, but the agent does not seem to be buying from the market import logging from datetime import datetime, timedelta import pandas as pd from dateutil import rrule as rr from assume import World from assume.common.forecasts import NaiveForecast from assume.common.market\_objects import MarketConfig, MarketProduct import dotenv import numpy as np import pandas as pd dotenv.load\_dotenv() from wondergrid.datasets import load\_dataset from wondergrid.datasets.geo import GeoDataset from wondergrid.datasets.dmk import DMKDataset from wondergrid.datasets.era5 import ERA5Dataset from wondergrid.simulator.weather import DataReplayWeatherModel from wondergrid.simulator.networkuser import DataReplayNetworkUserModel from wondergrid.simulator.core import Simulation from ResidentAgent import ResidentAgent log = logging.getLogger(\_\_name\_\_) db\_uri = "sqlite:///local\_db/assume\_db.db" world = World(database\_uri=db\_uri) start = datetime(2022, 1, 1) end = datetime(2022, 12, 31) index = pd.date\_range( start=start, end=end + timedelta(hours=24), freq="h", ) sim\_id = "world\_script\_simulation" world.setup( start=start, end=end, # save\_frequency\_hours=48, save\_frequency\_hours=24, simulation\_id=sim\_id, #index=index, ) # Configuration of the energy market # Only one market marketConf = MarketConfig( market\_id="EOM", # Energy Only Market opening\_hours=rr.rrule(rr.HOURLY, interval=24, dtstart=start, until=end), opening\_duration=timedelta(hours=1), market\_mechanism="pay\_as\_clear", market\_products=[MarketProduct(timedelta(hours=1), 24, timedelta(hours=1))], maximum\_bid\_volume=20000, maximum\_bid\_price=100, additional\_fields=["block\_id", "link", "exclusive\_id"], ) mo\_id = "market\_operator" world.add\_market\_operator(id=mo\_id) world.add\_market(market\_operator\_id=mo\_id, market\_config=marketConf) # Initialize the dataset dmkdataset: DMKDataset = load\_dataset('fluvius/dmk') dmkdataset = dmkdataset.filter(n=1) # Initialize the agent residence = ResidentAgent() # load the agent with the first profile (draft) for (iid, profile, metadata) in dmkdataset.get\_profiles(): profile = profile.resample('h').mean() residence.set\_load(profile) residenceAgent = { "id": "agent1", "data": residence.get\_load(), "role": "consumer" } # Get panda Series of timestamp and load agentDemand = residence.get\_load()['load'] print(agentDemand) world.add\_unit\_operator("demand\_operator") # Link demand list with forecaster (failed) demand\_forecast = NaiveForecast(index, demand=agentDemand) world.add\_unit( id="agent1", unit\_type="demand", unit\_operator\_id="demand\_operator", unit\_params={ "min\_power": 0, "max\_power": 4000, "bidding\_strategies": {"EOM": "naive\_eom"}, "technology": "demand", }, forecaster=demand\_forecast, ) world.add\_unit\_operator("unit\_operator") nuclear\_forecast = NaiveForecast(index, availability=1, fuel\_price=3, co2\_price=0.1) world.add\_unit( id="nuclear\_unit", unit\_type="power\_plant", unit\_operator\_id="unit\_operator", unit\_params={ "min\_power": 200, "max\_power": 5000, "bidding\_strategies": {"EOM": "naive\_eom"}, "technology": "nuclear", }, forecaster=nuclear\_forecast, ) # Number of agents num\_agents = 10000 # Run simulation world.run() |  |
| March | ChatGPT | start = datetime(2022, 1, 1) end = datetime(2022, 12, 31) index = pd.date\_range( start=start, end=end + timedelta(hours=24), freq="h", ) sim\_id = "world\_script\_simulation" print(index) the index is one hour too long |  |
| March | ChatGPT | no that not fix it |  |
| March | ChatGPT | how to time how long a line of code runs |  |
| March | ChatGPT | import logging import random import time from datetime import datetime, timedelta import pandas from dateutil import rrule as rr import pandas as pd from assume import World from assume.common.forecasts import NaiveForecast from assume.common.market\_objects import MarketConfig, MarketProduct import dotenv dotenv.load\_dotenv() from wondergrid.datasets import load\_dataset from wondergrid.datasets.dmk import DMKDataset log = logging.getLogger(\_\_name\_\_) db\_uri = "sqlite:///local\_db/assume\_db.db" world = World(database\_uri=db\_uri) start = datetime(2022, 1, 1) end = datetime(2022, 12, 31, 23, 45, 00) index = pd.date\_range( start=start, end=end, freq="15min", ) sim\_id = "sim" world.setup( start=start, end=end, save\_frequency\_hours=24, simulation\_id=sim\_id, index=index, ) # Configuration of the energy market # Only one market marketConf = MarketConfig( market\_id="EOM", # Energy Only Market opening\_hours=rr.rrule(rr.HOURLY, interval=24, dtstart=start, until=end), opening\_duration=timedelta(hours=1), market\_mechanism="pay\_as\_clear", market\_products=[MarketProduct(timedelta(hours=1), 24, timedelta(hours=1))], maximum\_bid\_volume=20000, # choose the value wisely maximum\_bid\_price=15000, additional\_fields=["block\_id", "link", "exclusive\_id"], ) mo\_id = "market\_operator" world.add\_market\_operator(id=mo\_id) world.add\_market(market\_operator\_id=mo\_id, market\_config=marketConf) # Setting up agent0 # Load the CSV into a DataFrame, ensuring the datetime column is parsed df = pd.read\_csv('MeasuredForecastedLoadAgent0.csv') # Remove unnecessary columns columns\_to\_remove = ['Datetime','Resolution code', 'Most recent P10', 'Most recent P90', 'Day-ahead 6PM forecast', 'Day-ahead 6PM P10', 'Day-ahead 6PM P90', 'Most recent forecast', 'Week-ahead forecast'] df = df.drop(columns=columns\_to\_remove, errors='ignore') # 'errors=ignore' prevents errors if a column is missing # Reverse time order (starts with 31/12/2022) df = df.apply(lambda col: col[::-1].values) # Set index to the index of the simulation df = df.set\_index(index) print(type(df)) # Initialize the dataset for residential units dmkdataset: DMKDataset = load\_dataset('fluvius/dmk') # We will work with 5 sets for now dmkdataset = dmkdataset.filter(n=5) profile = pandas.DataFrame() # Make a list of sets to randomly choose from loads = [] feeds = [] # load the agent with the profile for (iid, profile, metadata) in dmkdataset.get\_profiles(): loads.append(profile['load']) feeds.append(profile['feedin']) #print(loads) #print(feeds) # Set up load unit world.add\_unit\_operator("agent0") # Link load list with forecaster load\_forecast = NaiveForecast(index, demand=df) world.add\_unit( id="demand\_unit", # YOU CANNOT CHANGE THE ID TO ANYTHING OTHER THAN DEMAND OR TYPE OF POWER UNIT unit\_type="demand", unit\_operator\_id="agent0", unit\_params={ "min\_power": 0, "max\_power": 10000, "bidding\_strategies": {"EOM": "naive\_eom"}, "technology": "demand", }, forecaster=load\_forecast, ) # Set up load unit world.add\_unit\_operator("load\_operator") # Link load list with forecaster load\_forecast = NaiveForecast(index, demand=random.choice(loads)) world.add\_unit( id="demand\_unit", # YOU CANNOT CHANGE THE ID TO ANYTHING OTHER THAN DEMAND OR TYPE OF POWER UNIT unit\_type="demand", unit\_operator\_id="load\_operator", unit\_params={ "min\_power": 0, "max\_power": 10000, "bidding\_strategies": {"EOM": "naive\_eom"}, "technology": "demand", }, forecaster=load\_forecast, ) # Set up feedin unit as producer unit world.add\_unit\_operator("feedin\_operator") feedin\_forecast = NaiveForecast(index, availability=1, fuel\_price=3, co2\_price=0.1, demand=random.choice(feeds)) world.add\_unit( id="nuclear\_unit", unit\_type="power\_plant", unit\_operator\_id="feedin\_operator", unit\_params={ "min\_power": 100, "max\_power": 1000, "bidding\_strategies": {"EOM": "naive\_eom"}, "technology": "solar", }, forecaster=feedin\_forecast, ) # Set up producer unit world.add\_unit\_operator("power\_operator") nuclear\_forecast = NaiveForecast(index, availability=1, fuel\_price=3, co2\_price=0.1) world.add\_unit( id="nuclear\_unit", unit\_type="power\_plant", unit\_operator\_id="power\_operator", unit\_params={ "min\_power": 100, "max\_power": 1000, "bidding\_strategies": {"EOM": "naive\_eom"}, "technology": "nuclear", }, forecaster=nuclear\_forecast, ) # Time the simulation start\_time = time.perf\_counter() # Run simulation world.run() end\_time = time.perf\_counter() print(f"Execution time of simulation: {end\_time - start\_time:.6f} seconds") # timeit # python profilers time # tracemalloc Traceback (most recent call last): File "C:\Users\samee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10\_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\mango\util\scheduling.py", line 31, in \_raise\_exceptions raise fut.exception() File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10\_3.10.3056.0\_x64\_\_qbz5n2kfra8p0\lib\asyncio\tasks.py", line 232, in \_\_step result = coro.send(None) File "C:\Users\samee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10\_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\mango\util\scheduling.py", line 211, in run return await self.\_coro File "C:\Users\samee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10\_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\assume\common\units\_operator.py", line 416, in submit\_bids orderbook = await self.formulate\_bids( File "C:\Users\samee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10\_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\assume\common\units\_operator.py", line 483, in formulate\_bids product\_bids = unit.calculate\_bids( File "C:\Users\samee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10\_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\assume\common\base.py", line 106, in calculate\_bids bids = self.bidding\_strategies[market\_config.market\_id].calculate\_bids( File "C:\Users\samee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10\_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\assume\strategies\naive\_strategies.py", line 59, in calculate\_bids volume = unit.calculate\_ramp( File "C:\Users\samee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10\_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\assume\common\base.py", line 372, in calculate\_ramp if power > 0 and op\_time < 0 and op\_time > -self.min\_down\_time: ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all() ERROR:mango.util.scheduling:got exception in scheduled event |  |
| March |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |