Robo de negociações

Aprendizado do curso gratuito de linguagem Python pela Data Science Academy (https://www.datascienceacademy.com.br/cursosgratuitos)

Essa é uma implementação simples de um algoritmo de aprendizado por reforço (Q-learning) aplicado ao problema de negociação de ações.

> Q-learning é uma abordagem de aprendizado por reforço que permite a um agente aprender a melhor estratégia em um ambiente interativo, como um jogo, através de tentativa e erro, mantendo uma tabela de valores associados a diferentes ações em diferentes situações.

In [1]:
```
from platform import python_version
print('Versão da linguagem Python utilizada: ', python_version())
```

```
Versão da linguagem Python utilizada:  3.11.5
```

Importando bibliotecas

In [ ]:
```
#!pip install -q plotly yfinance
```

In [2]:
```
import yfinance as yf
import random
import plotly.graph_objects as go
import pandas as pd
import numpy as np
from datetime import datetime
```

Obtenção de dados

In [3]:
```
ticker = 'PETR4.SA'
petro = yf.download(ticker, start='2018-01-01', end='2023-12-31')
```

```
[**********100%%**********]  1 of 1 completed
```

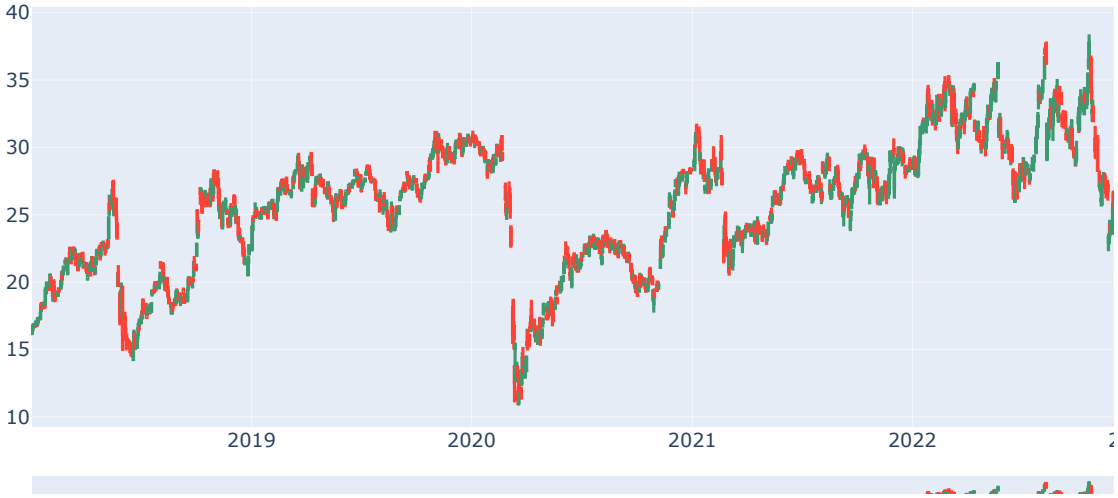
Visualização dos dados

In [4]:
```
dados = pd.DataFrame(petro)
dados.head(3)
```

Out[4]:

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-02	16.190001	16.549999	16.190001	16.549999	5.651068	33461800
2018-01-03	16.490000	16.719999	16.370001	16.700001	5.702286	55940900
2018-01-04	16.780001	16.959999	16.620001	16.730000	5.712530	37064900

In [5]:
```
fig = go.Figure(
    data=[go.Candlestick(x=dados.index,
                         open=dados['Open'],
                         high=dados['High'],
                         low=dados['Low'],
                         close=dados['Close']
                         )])
fig.show()
```



Configurações

Hiperparâmetros

In [6]:
```
episodios = 1000
alfa = 0.1  # Taxa de aprendizado
gama = 0.99  # Taxa de recompensa
epsilon = 0.1  # Exploração vs Exploração
```

Ambiente de negociação

In [7]:
```
precos = dados.Close.values
acoes = ['comprar', 'vender','manter'] # Ações de negociação
saldo_inicial = 1000
num_acoes_inicial = 0
```

Função de execução de Ações

In [8]:
```
# executar os passos do robo trading
def executar_acao (estado, acao, saldo, num_acoes, preco):

    # comprar
    if acao == 0:
        if saldo >= preco:
            num_acoes += 1
            saldo -= preco

    # vender
    elif acao == 1:
        if num_acoes > 0:
            num_acoes -= 1
            saldo += preco

    # lucro
    lucro = saldo + num_acoes*preco - saldo_inicial

    return (saldo, num_acoes, lucro)
```

Algoritmo Q-learning

Inicialização da tabela Q

- Matriz iniciada em '0' que sera atualizada durante o treinamento
- len(precos) representa o numero de estados
- len(acoes) as ações da função 'executar_acao()'

In [9]:
```
q_tabela = np.zeros((len(precos), len(acoes)))
```

Treinamento do algoritmo

- O treinamento é realizado por meio de vários episódios, onde um episódio representa uma iteração completa sobre o conjunto de dados históricos.
- No início de cada episódio, o saldo e o número de ações são reinicializados.
- Dentro de cada episódio, o algoritmo percorre os dados históricos de preço.
- O agente escolhe a próxima ação com base em uma política epsilon-greedy.
- A ação escolhida é executada, e a tabela Q é atualizada com base na recompensa obtida.

In [10]:
```
for _ in range(episodios):

  saldo = saldo_inicial
  num_acoes = num_acoes_inicial

  for i, preco in enumerate(precos[:-1]):
    estado = i

    if np.random.random() < epsilon:
      acao = random.choice(range(len(acoes)))
    else:
      acao = np.argmax(q_tabela[estado])

    saldo, num_acoes, lucro = executar_acao(estado, acao, saldo, num_acoes,preco)
    prox_estado = i + 1

    q_tabela[estado][acao] += alfa * (lucro + gama * np.max(q_tabela[prox_estado]) - q_tabela[estad
print('Treinamento concluído')
```

```
Treinamento concluído
```

Execução do algoritmo treinado

- Antes de executar o algoritmo treinado, o saldo e o número de ações são reinicializados.
- O código percorre novamente os dados históricos de preços executando o modelo treinado.
- O agente escolhe a ação com o maior valor na tabela Q para o estado atual.
- Ao final, o saldo final e o lucro obtido são calculados.

In [11]:
```
saldo = saldo_inicial
num_acoes = num_acoes_inicial

for i, preco in enumerate(precos[:-1]):
  estado = iacao = np.argmax(q_tabela[estado])
  saldo, num_acoes, _ = executar_acao(estado, acao, saldo, num_acoes, preco)

print('Execução concluída')
```

```
Execução concluída
```

Resultados

In [12]:
```
print(f'\nO modelo treinado está acumulando um total de: {num_acoes} ações com ticker {ticker}')
print(f'\nÚltimo preço de fechamento: R${round(precos[-1], 2)}')
```

```
O modelo treinado está acumulando um total de: 51 ações com ticker PETR4.SA

Último preço de fechamento: R$37.24
```

Vendendo todas as ações no último preço de fechamento

In [13]:
```
saldo += num_acoes * precos[-1]
lucro = saldo - saldo_inicial
lucro_final = round(lucro, 2)
```

Relatório

In [14]:
```
print(f'\nRelatório de Negociação:')
print(f'\nSaldo inicial: {saldo_inicial}')
print(f'Saldo final: {round(saldo,2)}')
print(f'Lucro: {lucro_final}')
```

```
Relatório de Negociação:

Saldo inicial: 1000
Saldo final: 1900.44
Lucro: 900.44
```

Exportação de dados para csv

In [15]:
```
dados.to_csv('data/PETR4.csv', index=False)
```