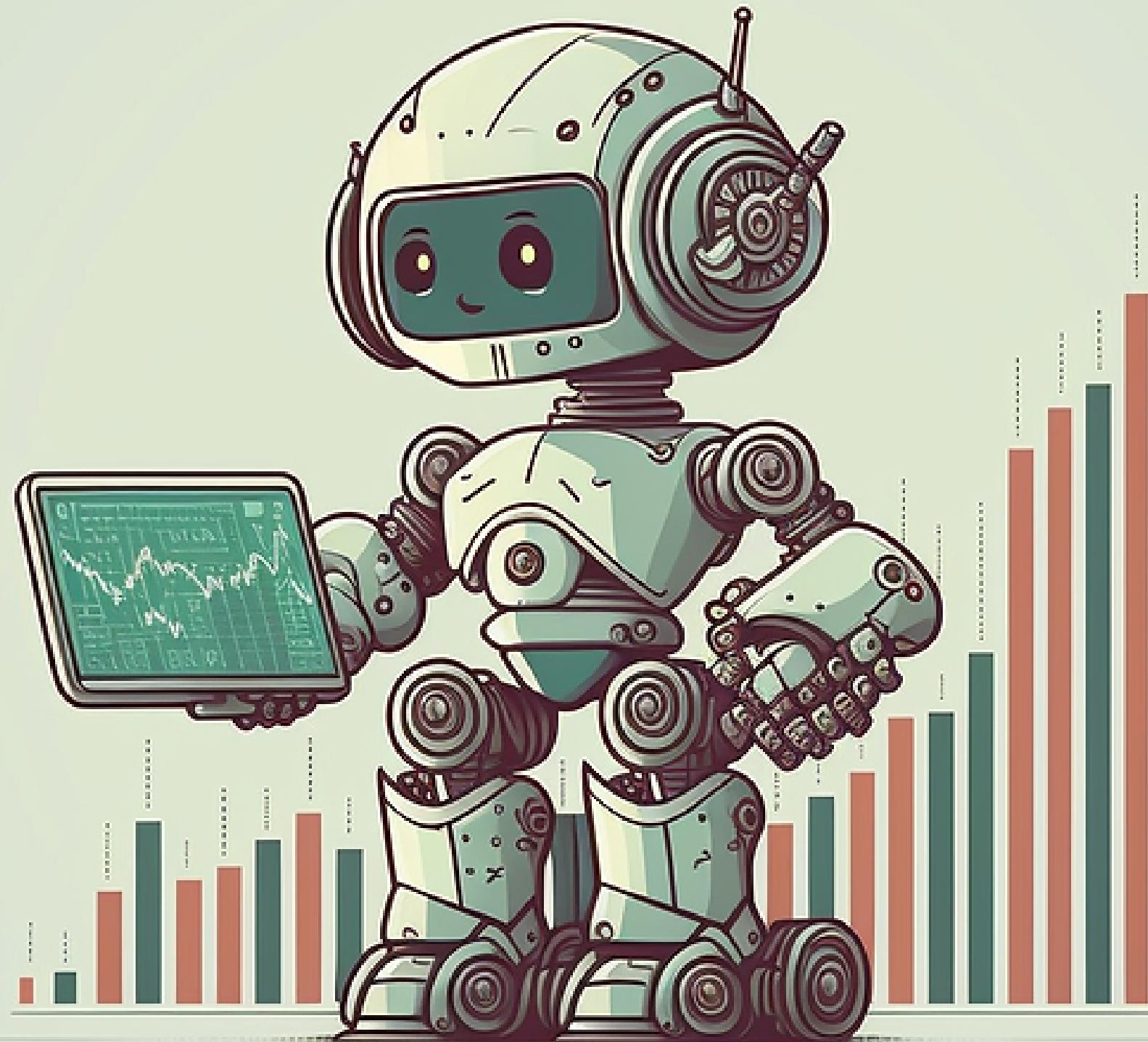


Aplicando Algoritmos de *Machine Learning* em *Python* para previsões no Mercado Financeiro

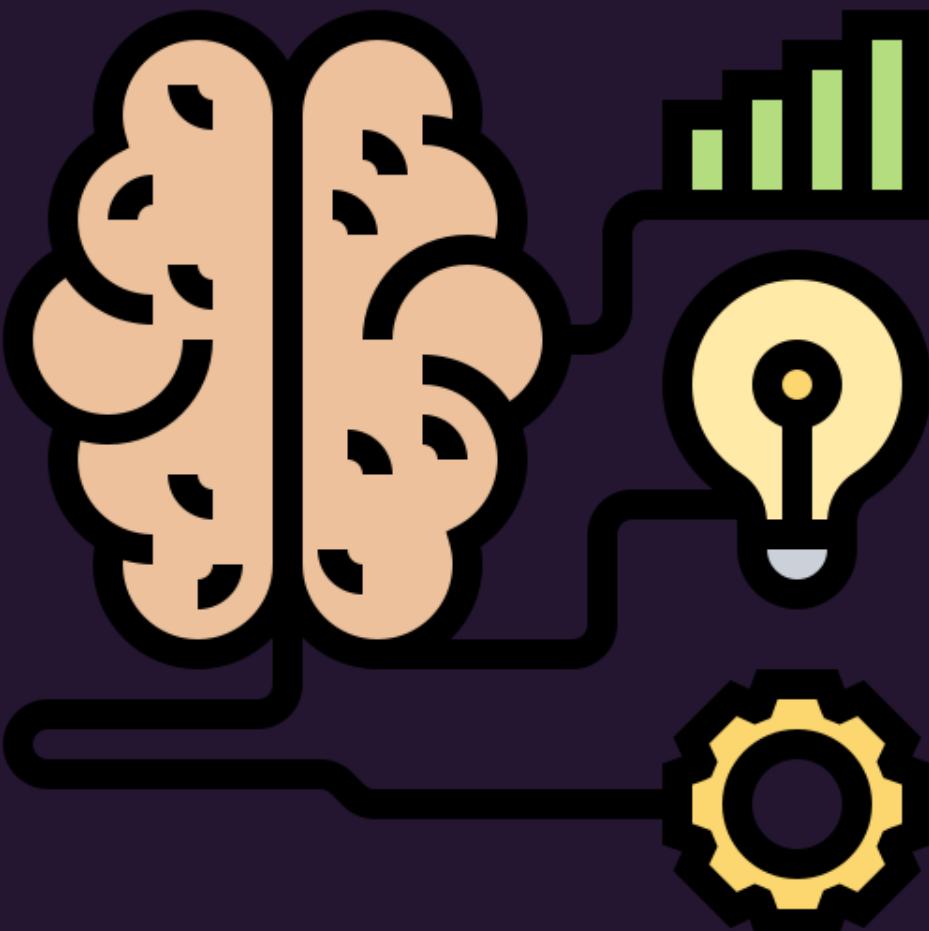
Cristian Andre Sanches, Guido Margonar Moreira e
Joao Pedro Padoan



Introdução

Algoritmos de *Machine Learning* possuem uma extensa gama de métodos capazes de realizar previsões baseadas nos dados utilizados em seu treinamento.

Utilizando esse conceito, podemos aplicar para um ativo no mercado financeiro, usando dados de cotação, abertura e fechamento, e modelando a fim de prever seu próximo valor.



Pontos Importantes para realização do projeto?

1 Preço das ações e moedas

As cotações estão em constante atualização. Dependendo da hora em que uma transação é feita, determinará se será positivo ou não. Lidar com essas mudanças é um desafio aos acionistas, e são dependentes de muitos fatores.

2 Importância do mercado financeiro

As flutuações no mercado financeiro são características da oferta e demanda. Isto é a base para o desenvolvimento empresarial, por isso prever seu comportamento seria vantajoso.

3 Algoritmos de *Machine Learning*

Os algoritmos de *Machine Learning*, como os vistos em sala, podem ser uma ferramenta poderosa para auxiliar nessas previsões.

Objetivo do Projeto

Este projeto tem por objetivo explorar como alguns dos paradigmas de *Machine Learning* podem nos ajudar na previsão dos preços de ações ou moedas, reconhecendo padrões ou características cujo o homem não conseguiria notar.

Apesar da complexidade envolvida na variação dos preços de mercado, para este projeto foram consideradas apenas as variações no preço do dollar em relação ao Yen ao longo de um dado período de tempo.



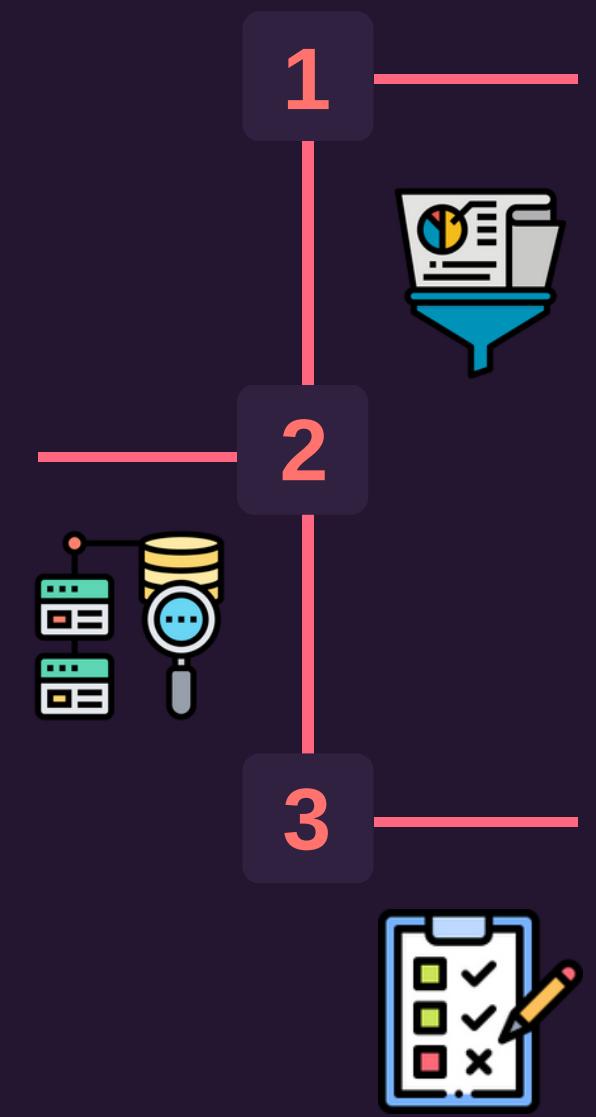
Metodologia



Aplicação de um modelo de *Machine Learning*

Divisão dos dados entre treino e teste

Escolhemos uma porcentagem da nossa base de dados para treinar os modelos de *Machine Learning* enquanto o restante é reservado para testes.



1 — Coleta e Pré-processamento de dados

Importamos dados do *dataset* e selecionamos os parâmetros relevantes para previsão e normalização dos valores.

3 — Avaliação e Refinamento do Modelo

Usamos o modelo para realizar as previsões, analisar os erros e ajustar os parâmetros para otimizar os resultados.

Datasets: Yfinance e MetaTrader5



Yfinance

O *Yahoo Finance* é um site que contém registros sobre a variação do preço de diversas empresas ao longo dos anos. Para usar os dados no *Python* foi usada a biblioteca *yfinance* que permite coletar várias informações com facilidade.

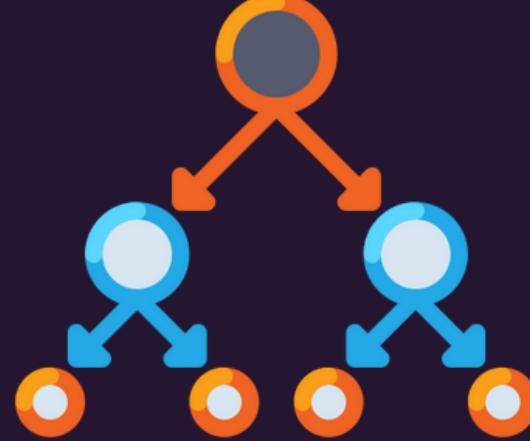


MetaTrader 5

MetaTrader5

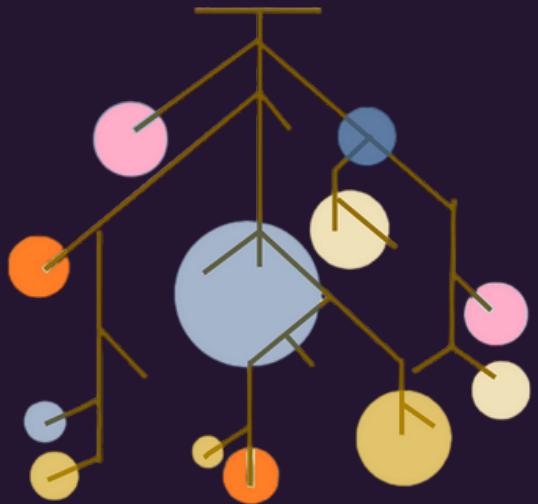
Como plataforma para obtenção dos dados do mercado financeiro, utilizamos a plataforma de negociação *Meta Trader 5* por conta da grande quantidade de dados oferecidos de forma gratuita e para acessá-los utilizamos a biblioteca própria no *Python* chamada *MetaTrader5*.

Algoritmos de *Machine Learning* que utilizamos



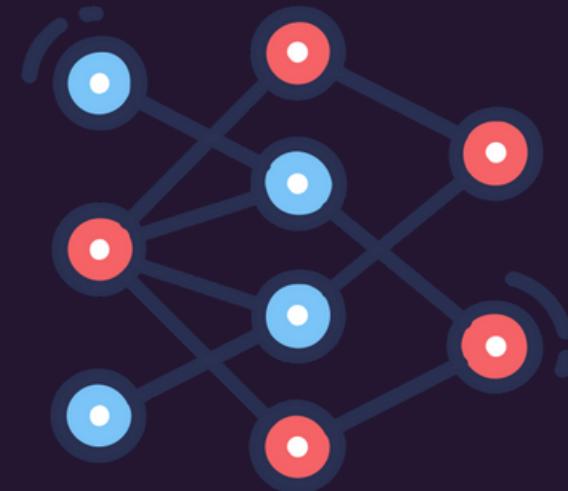
Árvores de Decisão

Uma árvore de decisão começa com a raiz contendo todos os dados, que são então divididos de acordo com suas características. Assim, cada nó seria uma pergunta/teste sobre o dado, sendo que os ramos seriam a resposta ao teste.



Random Forest

A regressão por *Random Forest* é um algoritmo de aprendizado de máquina que combina os princípios de aprendizagem em conjunto e árvores de decisão para tarefas de regressão.

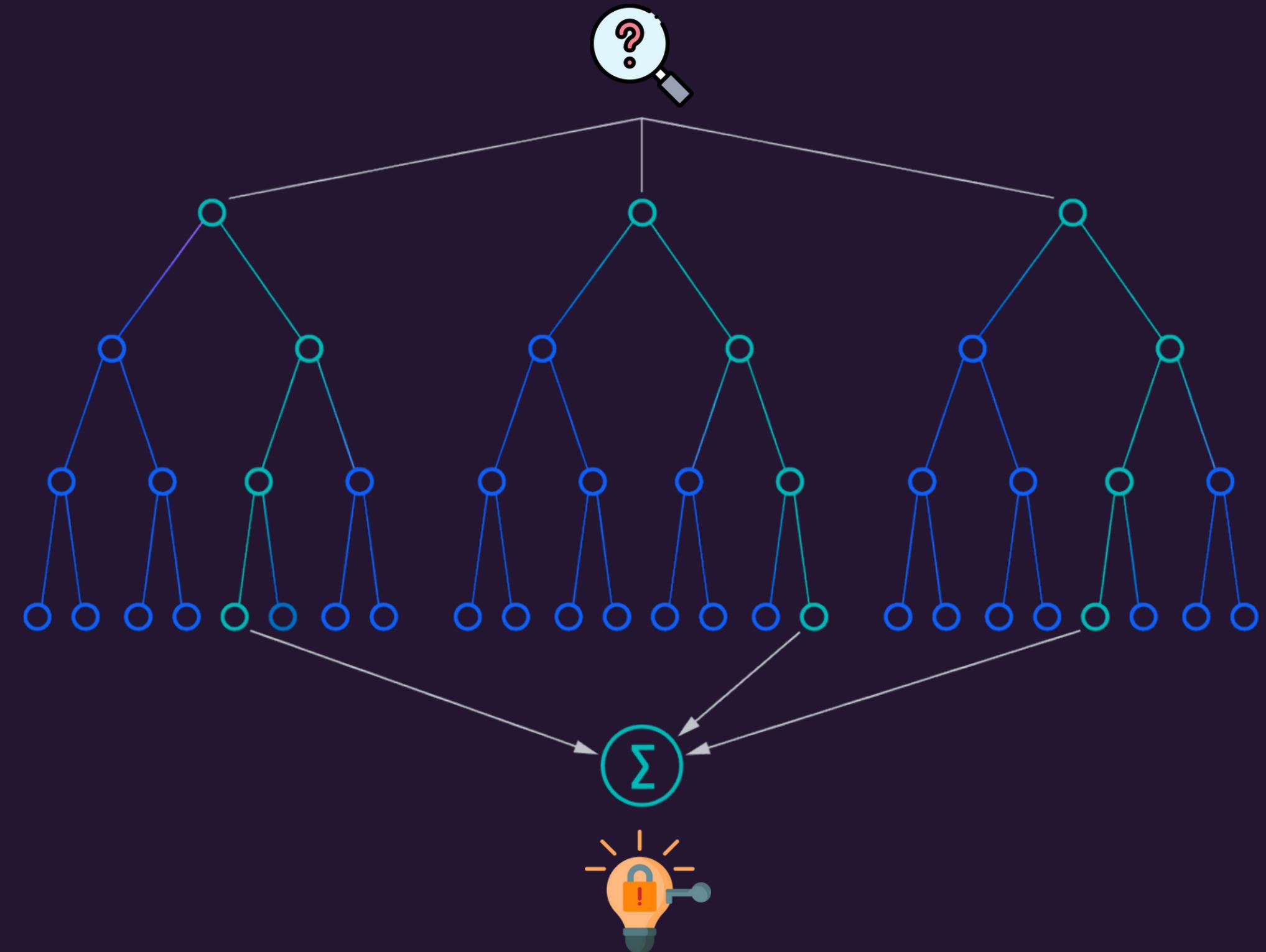


Redes Neurais

Uma rede neural é um modelo computacional inspirado na estrutura e funcionamento das redes neurais biológicas no cérebro humano.

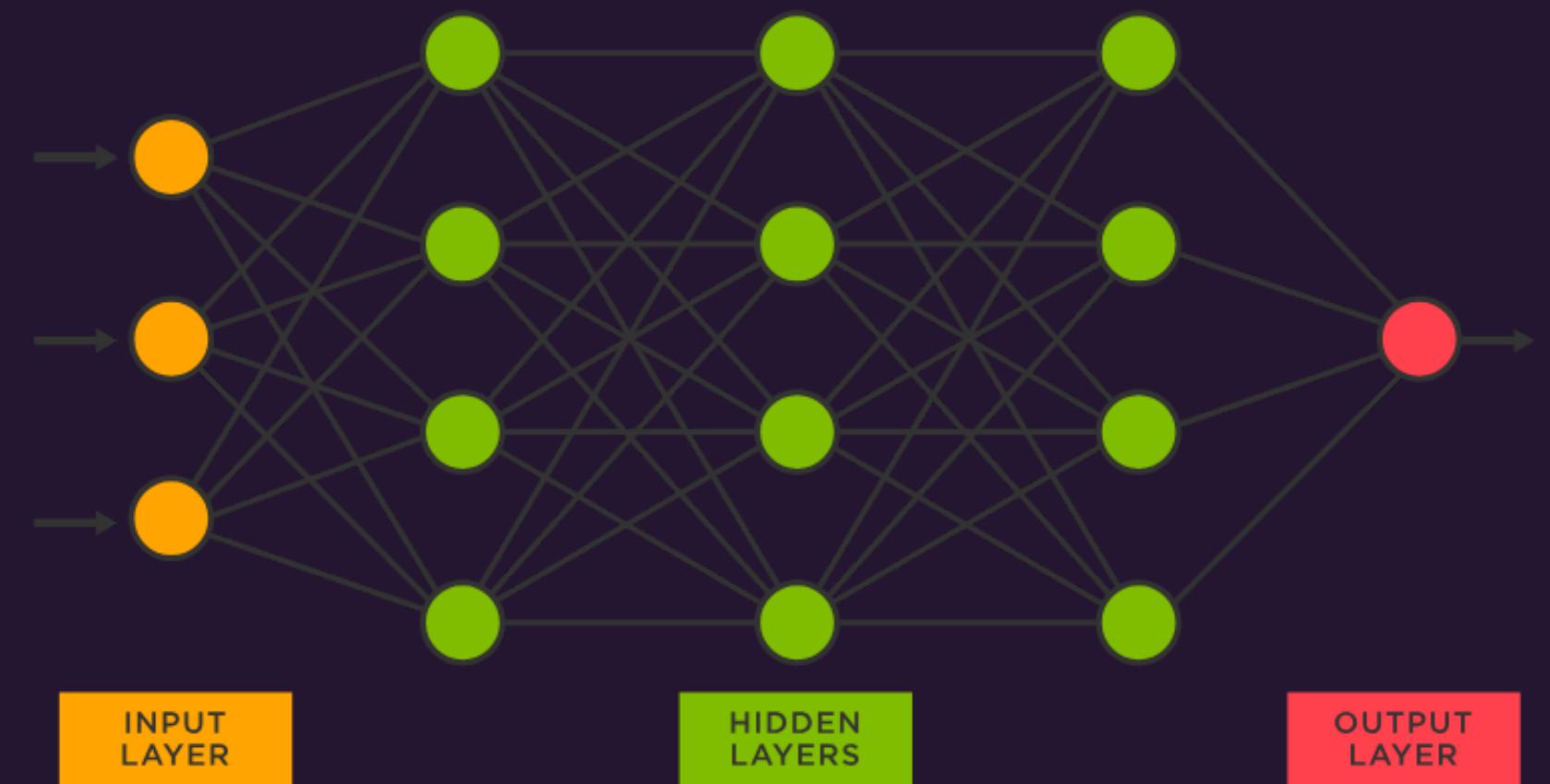
Random Forest Regression

- Coleção de árvores aleatórias.
- Treinamento das árvores.
- Média do resultado das árvores.



Redes Neurais

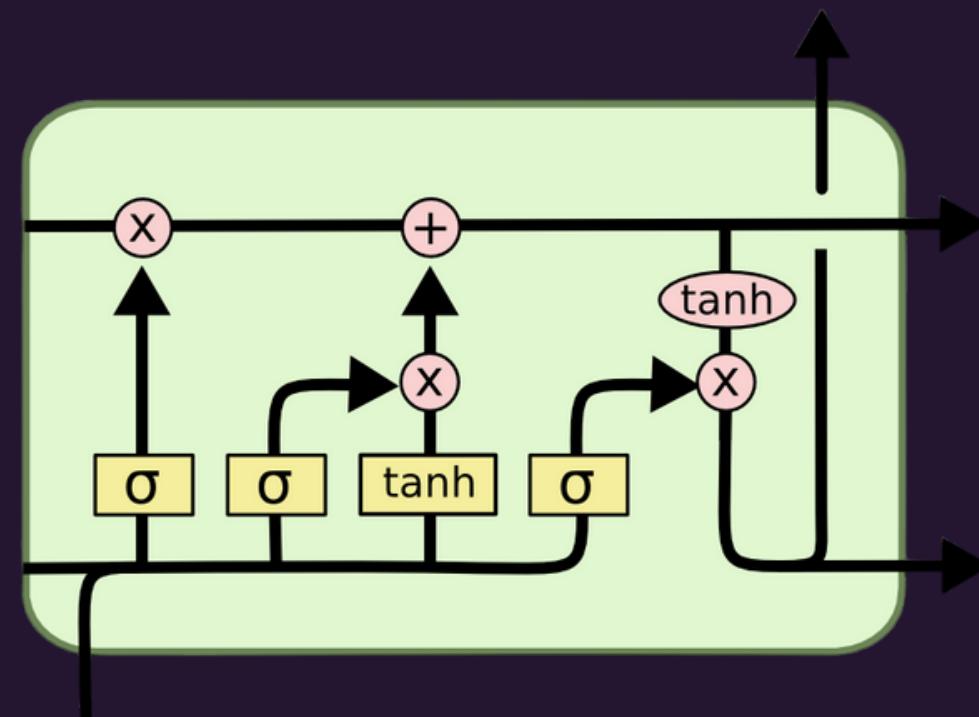
- Inspirado no cérebro.
- Camadas de neurônios com pesos.
- Abstrai bastante o significado dos dados.
- Utiliza outro algoritmo para definir pesos.



Redes Neurais

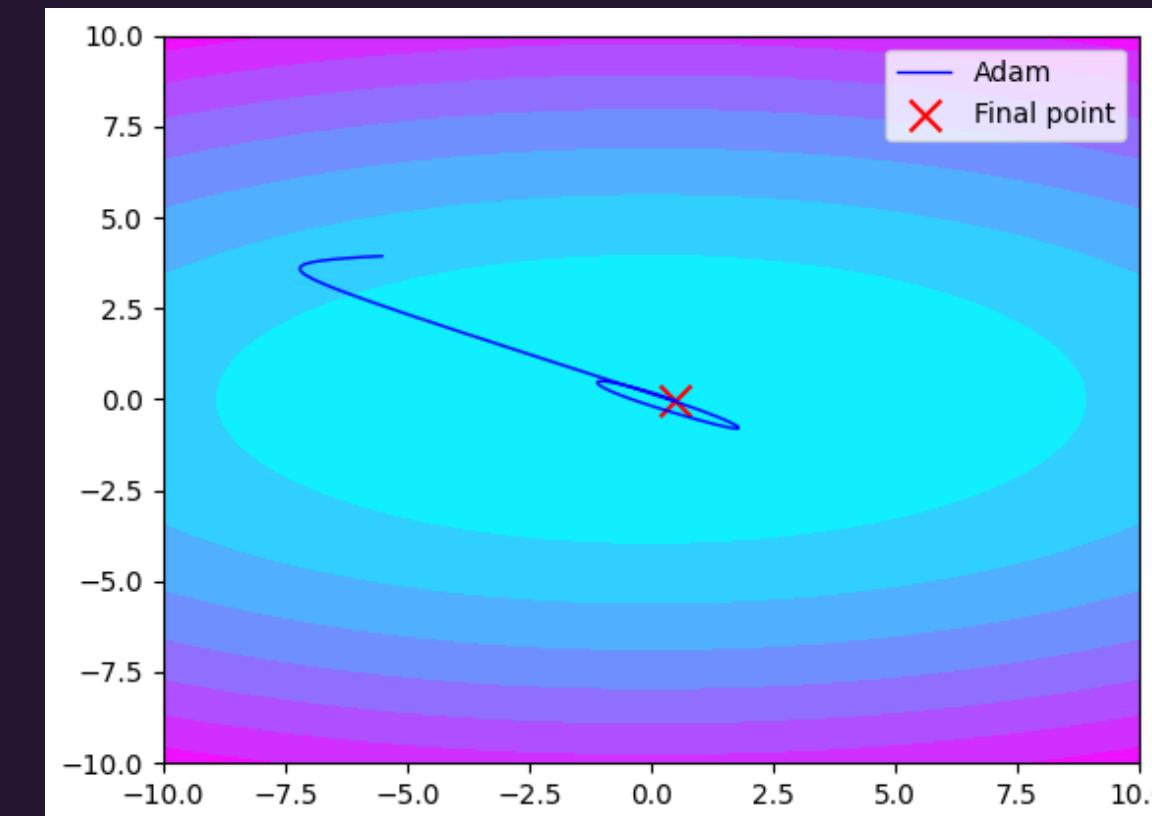
LSTM

O LSTM (*Long Short-Term Memory*) é um tipo de rede neural recorrente que é projetada para lidar com problemas de dependência de longo prazo em sequências de dados.



Sequential ADAM

O *Sequential ADAM* é um algoritmo de otimização que ajusta gradualmente os pesos e os parâmetros de um modelo de rede neural de maneira iterativa usando o método do gradiente descendente e a adaptação de momentos.



Outras Bibliotecas Importantes



Scikit Learn

O *sklearn* fornece métodos para os diferentes estágios da análise de dados, isto é: o pré-processamento e tratamento dos dados, a sua classificação, regressão, clusterização, redução de dimensionalidade e ajuste de parâmetros.



Tensorflow

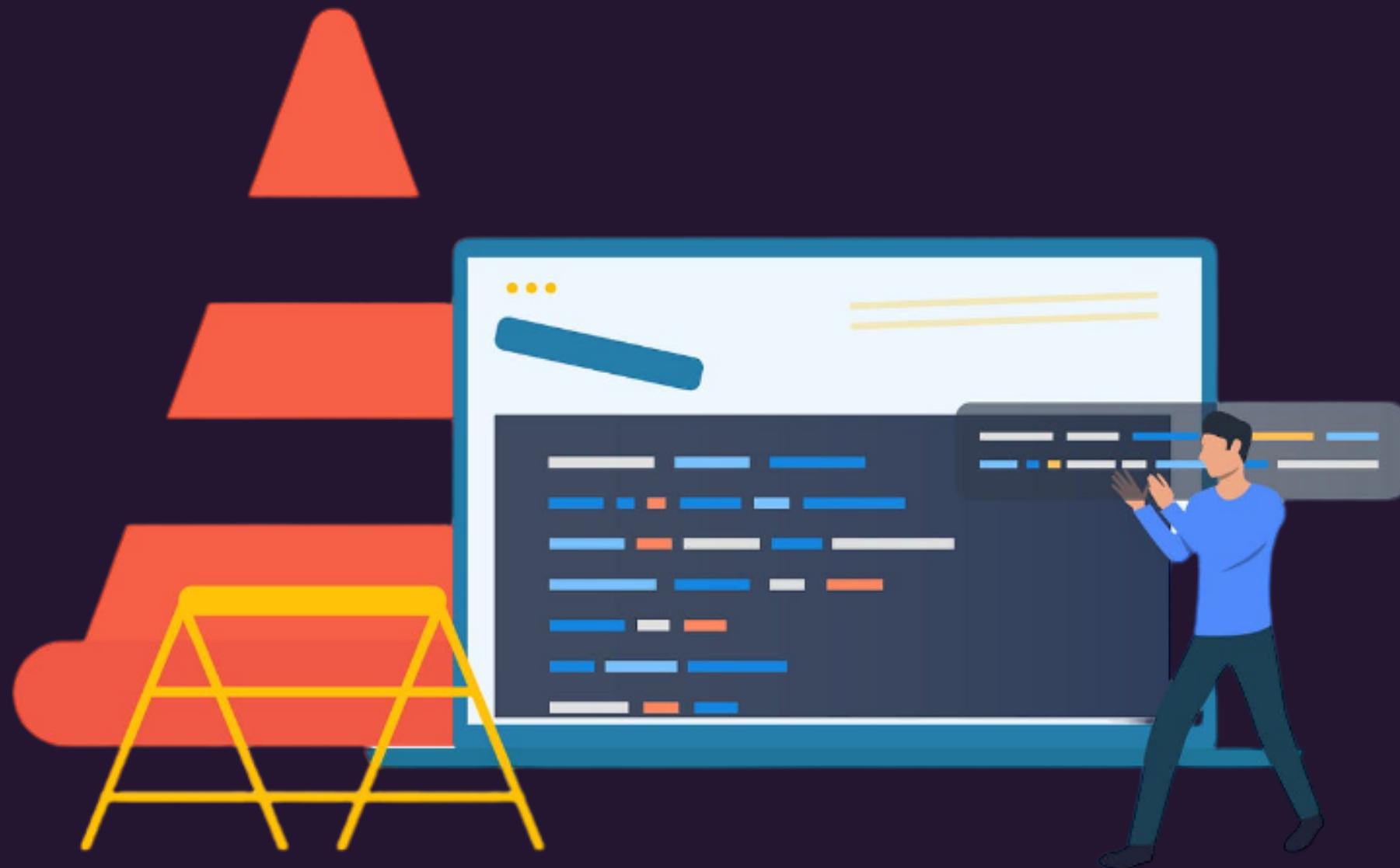
O *TensorFlow* é uma biblioteca open source para aplicações de machine learning e deep learning desenvolvida pela Google Brain, amplamente usada nas pesquisas e na indústria.



Keras

O *Keras* é uma API de redes neurais de código aberto de alto nível escrita em Python. Ela é desenvolvida a partir do *TensorFlow* e fornece uma interface amigável e interativa para desenvolver e treinar redes neurais.

Construção do Código



Carregamento dos dados

Rede Neural

```
if (mt5.initialize()): #pegar dados do Metatrader
    print("ok")
else:
    print("falha")
a = mt5.terminal_info()
ativo = "USDJPY" #ativo do dolar e japao
mt5.symbol_select(ativo, True)
f = mt5.copy_rates_from_pos(ativo, mt5.TIMEFRAME_M5, 0, 3000)
g = pd.DataFrame(f)
```

Random Forest

```
ticker = "AAPL"
start_date = "2018-01-01"
end_date = "2023-05-12"

data = yf.download(ticker,start= start_date,end=end_date)

df = pd.DataFrame(data)

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

Pré processamento

Rede Neural

```
#Tamanho de cada input para a rede neural  
Mrange = 30  
  
#Modelar os dados para treinamento  
threshold = 0.6  
data = g.filter(['close'])  
dataset = data.values  
training_data_len = math.ceil(len(dataset)*threshold)  
  
scaler = MinMaxScaler(feature_range=(-0.5,0.5))  
  
scale_data = scaler.fit_transform(dataset)
```

Random Forest

```
df.drop(['date','Volume'], axis=1,inplace=True)  
df.reset_index(drop=True,inplace=True)  
X = df[['Open','Close','High','Low','Adj Close']]#inputs  
y = df['Close']#Alvo
```

Divisão para dados de treino e teste

Rede Neural

```
train_data = scale_data[0:training_data_len,:]
x_train = []
y_train = []

for i in range(Mrange,len(train_data)):
    x_train.append(train_data[i-Mrange:i, 0])
    y_train.append(train_data[i, 0])

x_train,y_train = np.array(x_train),np.array(y_train)

x_train = np.reshape(x_train,(x_train.shape[0],Mrange,1))

test_data = scale_data[training_data_len - Mrange: , :]
x_test = []
y_test = dataset[training_data_len:,:]
for i in range(Mrange,len(test_data)):
    x_test.append(test_data[i-Mrange:i,0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

Random Forest

```
x_train,X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.3,random_state=1,shuffle=False)
```

Criação e Treinamento do modelo

Rede Neural

```
model = Sequential()  
model.add(LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))  
model.add(LSTM(100, return_sequences=False))  
model.add(Dense(25))  
model.add(Dense(1))  
  
model.compile(optimizer="sgd", loss='mean_squared_error')  
model.fit(x_train,y_train,batch_size=1,epochs=1)
```

Random Forest

```
rf = RandomForestRegressor(n_estimators=60,random_state=42)  
  
rf.fit(x_train, y_train)
```

Cálculo de Erro e Avaliação dos modelos

Rede Neural

```
prediction = model.predict(x_test)
prediction = scaler.inverse_transform(prediction)

rmse = np.sqrt(np.mean(prediction - y_test)*2)
print("Erro Quadrático médio {:.2f}%".format(rmse*100))

LancesCerto = 0
LancePerdidos = 0
listv = valid.values.tolist()

for i in range(1,len(prediction)):
    closeat = listv[i][0] # atual
    closeant = listv[i-1][0]# anterior
    prediat = listv[i][1] - rmse(listv[i][1])# previsão pro atual corrigida
    prediant = listv[i-1][1]- rmse(listv[i][1])# previsão anterior corrigida
    #Subiu
    if round(closeat,5) >= round(closeant, 5):
        if round(prediat, 5) >= round(prediant,5):#Previsão que subiria CERTA
            LancesCerto+=1
        else:#Previsão que desceria ou manteria Errada
            LancePerdidos+=1

    #Desceu
    else:
        if round(prediat, 5) <= round(prediant,5):#Previsão que desceria ou não mudaria CERTA
            LancesCerto += 1
        else: #Previsão que subiria ERRADA
            LancePerdidos += 1
```

Random Forest

```
y_pred = rf.predict(X_test)

mse = mean_squared_error(y_test,y_pred)

LancesCerto = 0
LancePerdidos = 0
listv = df['Close'].values.tolist()

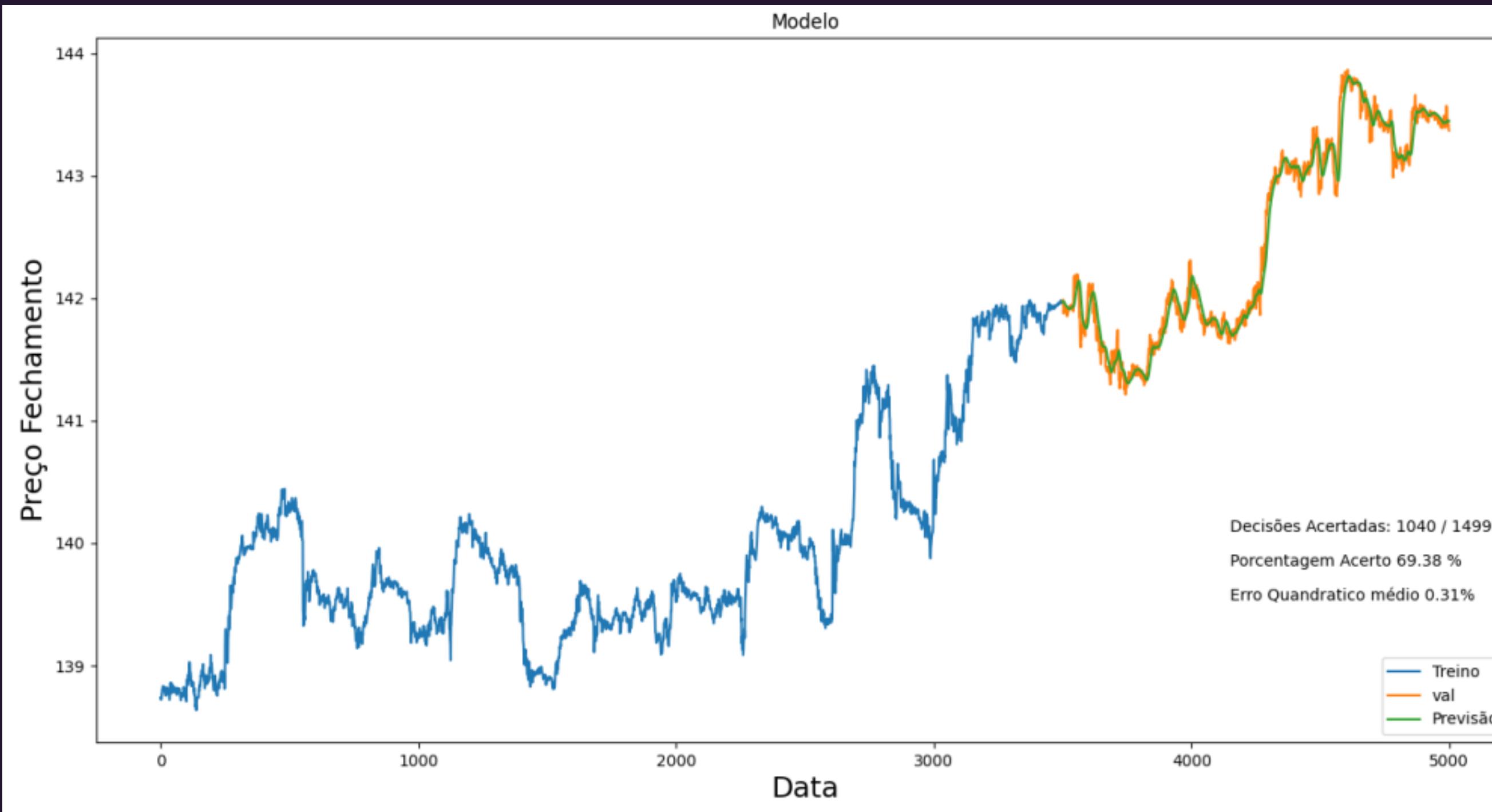
listv = listv[len(y_pred):]

for i in range(1,len(y_pred)):
    closeat = listv[i] # atual
    closeant = listv[i-1]# anterior
    prediat = y_pred[i] #- rmse# previsão pro atual
    prediant = y_pred[i-1]# previsão anterior
    #Subiu
    if round(closeat, 5) >= round(closeant, 5):
        if round(prediat, 5) >= round(prediant,5):#Previsão que subiria CERTA
            LancesCerto+=1
        else:#Previsão que desceria ou manteria Errada
            LancePerdidos+=1

    #desceu ou não mudou
    else:
        if round(prediat, 5) >= round(prediant,5):#Previsão que subiria ERRADA
            LancePerdidos += 1
        else: #Previsão que desceria ou não mudaria CERTA
            LancesCerto +=1
```

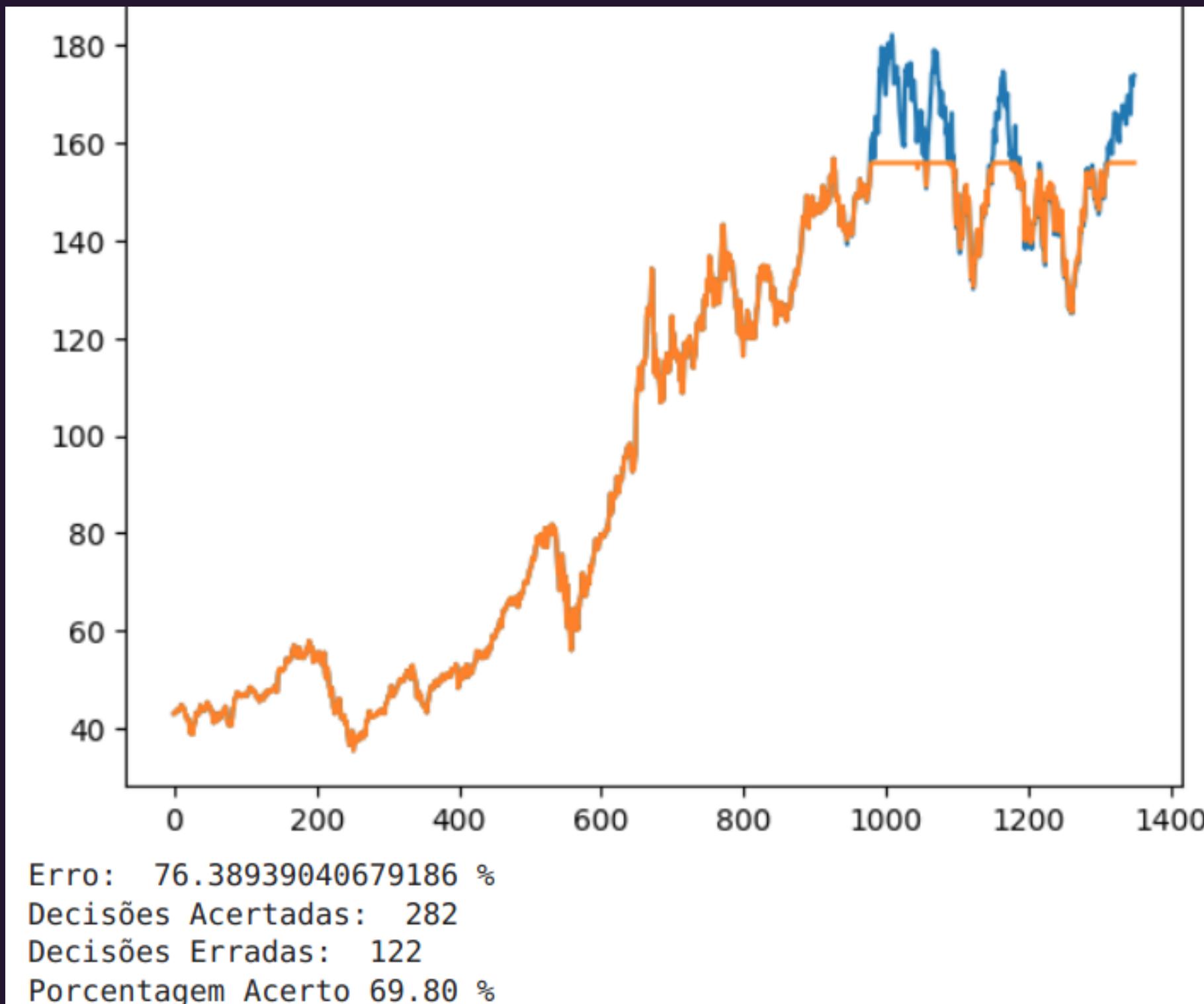
Visualização das previsões

Rede Neural



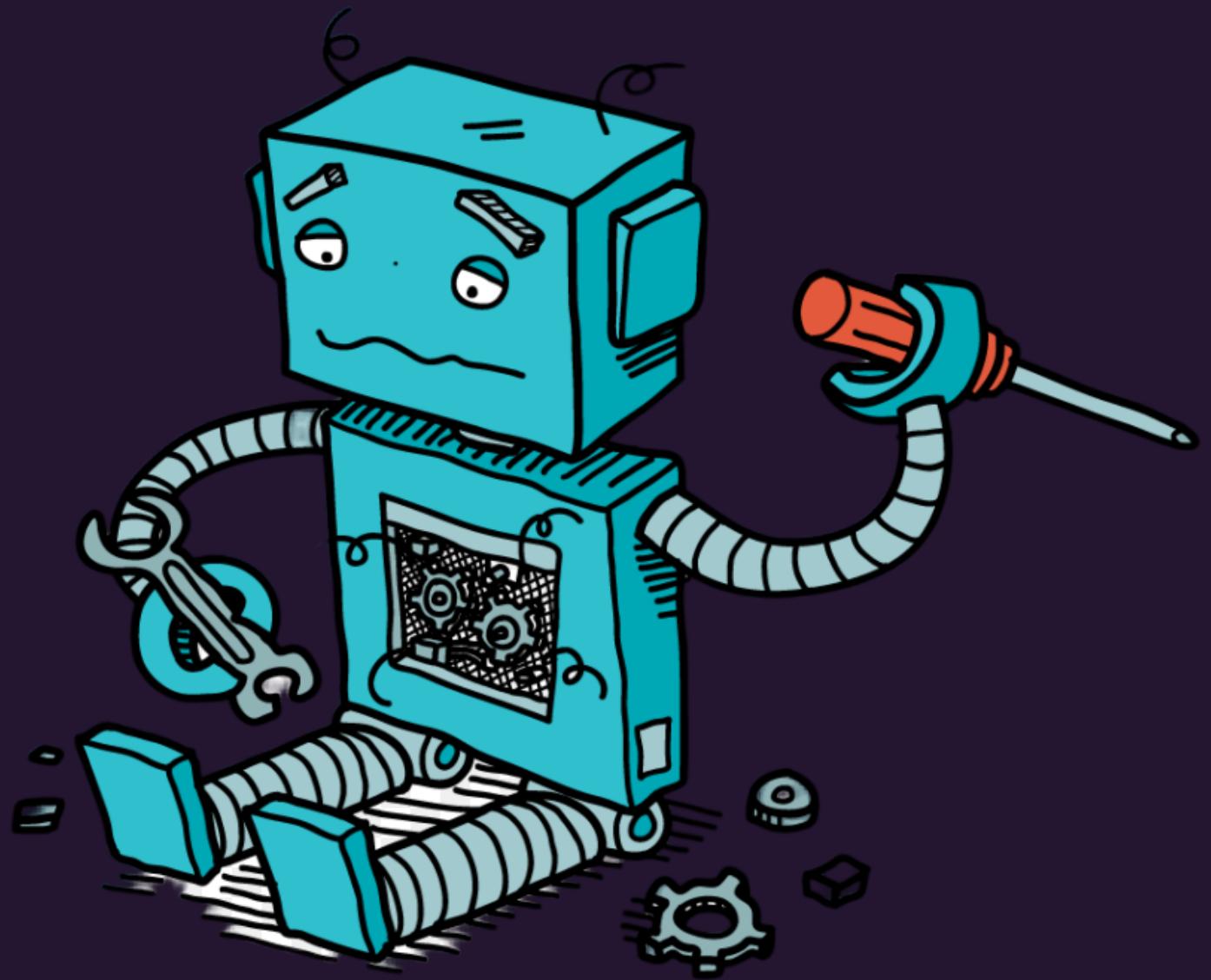
Visualização das previsões

Random Forest



Dificuldades e possíveis melhorias futuras

- Inclusão de mais fatores externos
- Refinamento e otimização dos modelos
- Volatilidade dos preços
- Atualização em tempo real
- Overfitting
- Ruído de dados



Conclusão

Após a análise dos testes realizados, concluímos que o algoritmo produziu bons resultados a partir dos dados obtidos, apresentando erros entre as verificações de subida e decida analisada pelas curvas da moeda.

Dessa forma os modelos estão longe de uma versão utilizável na prática, o que está dentro do esperado visto que nosso parâmetros de previsão foram apenas referentes a variação do valor no tempo.

Perguntas?