```
import pandas as pd
import numpy as np
import time
import datetime as dt
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# from google.colab import drive
# drive.mount('/content/drive')

# finance_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/finance.csv', encoding= 'unicode_escape'
, parse_dates=['Date'])
finance_df = pd.read_csv("../input/finance1/finance.csv", encoding= 'unicode_escape', parse_dates=['Dat
e'])
```

если в файле есть нестандартные символы (например, кириллица или спецсимволы).

столбец Date нужно интерпретировать как дату

```
X = finance_df
X['Date'] = finance_df['Date'].map(dt.datetime.toordinal)
```

Преобразуем дату в числовой формат

```
y = X.copy(deep=True).drop(labels='Date', axis=1).shift(-1).dropna()
```

Сдвигает данные на одну строку вверх

Удаляет последнюю строку

y — это значения на следующий день .мы хотим предсказать, какие будут значения завтра, на основе сегодняшних.

```
X.insert(0, 'Intercept', 1)
```

Добавляет столбец Intercept со значением 1 в начало X.
Нужен для свободного члена

```
X = X.iloc[0:-1]
y
```

Чтобы X и y имели одинаковое количество строк, удаляем последнюю строку X

	Adj Close AAPL	Volume AAPL	Adj Close ABC	Volume ABC	Adj Close AMZN	Volume AMZN	Adj Close BAC	Volume BAC	Adj Close BP	Volume BP	...
0	1.238824	4.204488e+09	11.963464	20746800.0	34.750000	29673700.0	32.568451	41283300.0	28.190603	20307100.0	...
1	1.321578	3.553878e+09	12.072557	18499200.0	34.160000	26026800.0	31.977856	55968800.0	28.584154	15497500.0	...
2	1.307426	2.055071e+09	12.122155	8516800.0	32.880001	23236100.0	31.131090	50533000.0	26.983763	15531900.0	...
3	1.257898	2.409145e+09	11.179917	59184000.0	34.009998	24386400.0	31.316097	59839000.0	27.442883	15717000.0	...
4	1.345573	2.708017e+09	11.471516	30875600.0	34.599998	24371600.0	31.792841	44185900.0	27.648401	16214000.0	...
...	...	...	...	...	...	...	...	...	...	...	...
830	136.555939	4.391332e+08	109.235848	7789100.0	3352.149902	25631900.0	32.369999	237647600.0	20.532003	146399000.0	...
831	135.369995	3.453017e+08	104.744514	5329400.0	3277.709961	13237600.0	33.369999	193834200.0	21.617310	95017100.0	...
832	129.869995	3.622987e+08	105.870003	3276200.0	3249.899902	13185300.0	34.540001	190095000.0	22.564486	61151200.0	...
833	121.260002	6.837847e+08	101.220001	4993900.0	3092.929932	19957700.0	34.709999	327837000.0	24.410000	110431100.0	...
834	121.260002	1.645600e+08	101.220001	1333167.0	3092.929932	4140953.0	34.709999	70933313.0	24.410000	22661449.0	...

835 rows × 40 columns

```
inv = lambda x: np.linalg.inv(x)
qr = lambda x: np.linalg.qr(x)
norm = lambda x: np.linalg.norm(x, 2)

start_time = time.time()
pseudo_inv1 = inv(X.T @ X) @ X.T
end_time = time.time()
time1 = end_time - start_time

b = pseudo_inv1 @ y

start_time = time.time()
Q, R = qr(X)
pseudo_inv2 = inv(R) @ Q.T
end_time = time.time()
time2 = end_time - start_time

error = norm(pseudo_inv1-pseudo_inv2)
print ("Sum of the squared error of difference between making pseudo_inv using the regular way and QR dec
omposition way:", error)
print ("Time for pseudo_inv using the regular way", time1)
print ("Time for pseudo_inv using the QR decomposition way", time2)

if time1 < time2:
  print("The regular way is faster.")
else:
  print("The QR decomposition way is faster.")
```

```
Sum of the squared error of difference between making pseudo_inv using the regular way and QR decompos
ition way: 6.854268592400155e-07
Time for pseudo_inv using the regular way 0.020267724990844727
Time for pseudo_inv using the QR decomposition way 0.014574289321899414
The QR decomposition way is faster.
```