An Algo Trading Strategy which made +8,371%: A Python Case Study

Backtesting of a simple breakout trading strategy with APIs and Python

Nikhil Adithyan · Follow
Published in Level Up Coding
7 min read · Nov 29, 2023

Photo by Behnam Norouzi on Unsplash

Introduction

There is literally no point in going after traditional algo trading strategies like SMA crossover or RSI threshold breakout strategy as it has proven to be obsolete given their simplistic nature and more importantly, the massive volume of participants who are trying to implement them in the market.

So instead of embracing those strategies, it's time to try something new. In this article, we'll be using Python and Benzinga's APIs to construct and backtest a new trading strategy that will help us beat the market.

With that being said, let's dive into the article!

The Trading Strategy

Before moving to the coding part, it's essential to have a good background on the strategy we're going to build in this article. Our trading strategy follows the principle of simplicity yet a very effective breakout strategy.

We enter the market if: the stock's current high exceeds the 50-week high

We exit the market if: the stock's current low sinks below the 40-week low

We'll be using the Donchian Channel indicator in order to keep track of the 50-week high and the 40-week low. This strategy is a weekly trading system, so, we'll be backtesting it on the weekly timeframe.

And, there you go! This is the strategy we're going to backtest in this article. As simple as it is, right?! Now, let's move on to the coding part of the article.

Importing Packages

In this article, we are going to use four primary packages which are `pandas`, `requests`, `pandas_ta` and `matplotlib`, and the secondary/optional packages include `termcolor` and `math`. The following code will import all the mentioned packages into our Python environment:

```
# IMPORTING PACKAGES

import pandas as pd
import requests
import pandas_ta as ta
import matplotlib.pyplot as plt
```

```
from termcolor import colored as cl
import math

plt.rcParams['figure.figsize'] = (20,10)
plt.style.use('fivethirtyeight')
```

If you haven't installed any of the imported packages, make sure to do so using the `pip` command in your terminal.

Extracting Historical Data

We are going to backtest our breakout strategy on Apple's stock. So in order to obtain the historical stock data of Apple, we are going to use Benzinga's Historical Bar Data API endpoint. The following Python code uses the endpoint to extract Apple's stock data from 1993:

```
# EXTRACTING HISTORICAL DATA

def get_historical_data(symbol, start_date, interval):
    url = "https://api.benzinga.com/api/v2/bars"
    querystring = {"token":"YOUR API KEY","symbols":f"{symbol}","from":f"{st

    hist_json = requests.get(url, params = querystring).json()
    df = pd.DataFrame(hist_json[0]['candles'])

    return df

aapl = get_historical_data('AAPL', '1993-01-01', '1W')
aapl.tail()
```

In the above code, we are defining a function named `get_historical_data` which takes the stock symbol, the starting date of the data, and the interval between the data points.

Inside the function, we are storing the API URL and the query strings into their respective variables. Make sure to replace `YOUR API KEY` with secret Benzinga API key which you can obtain after creating an account with them. Then, we are making an API call to obtain the data and converting the JSON response into a Pandas dataframe which we are returning at the end.

Using the defined function, we are extracting Apple's historical stock data from 1993 on a weekly timeframe. This is the end output:

	time	open	high	low	close	volume	dateTime
1607	1698019200000	170.91	174.010	165.67	168.22	286078100	2023-10-23T16:00:00.000-04:00
1608	1698624000000	169.02	177.780	167.90	176.65	310075880	2023-10-30T16:00:00.000-04:00
1609	1699228800000	176.38	186.565	176.21	186.40	303653020	2023-11-06T15:00:00.000-05:00
1610	1699833600000	185.82	190.960	184.21	189.69	262880720	2023-11-13T15:00:00.000-05:00
1611	1700438400000	189.89	192.930	189.25	189.97	148351450	2023-11-20T15:00:00.000-05:00

Apple's historical data (Image by Author)

Awesome, now let's move on to calculating the Donchian Channel indicator for the extracted historical data of Apple.

Donchian Channel Calculation

If we dive deep into the mathematics of the indicator, it will demand a separate article on its own for the explanations. So here's a general overview of the indicator. Basically, the Donchian Channel reveals the highest high and the lowest low of a stock over a specified period of time.

The following code uses `pandas_ta` for the calculation of the indicator:

```
# CALCULATING DONCHIAN CHANNEL

aapl[['dcl', 'dcm', 'dcu']] = aapl.ta.donchian(lower_length = 40, upper_leng
aapl = aapl.dropna().drop('time', axis = 1).rename(columns = {'dateTime':'da
aapl = aapl.set_index('date')
aapl.index = pd.to_datetime(aapl.index)

aapl.tail()
```

In the first line, we are using the `donchian` function provided by `pandas_ta` to calculate the indicator. The function takes two parameters: the lower length and the upper length which are the lookback periods for the lowest low and highest high respectively. We are mentioning the lower and upper lengths as 40 and 50

respectively since our strategy demands 40-week low and 50-week high.

After the calculation, we are performing some data manipulation tasks to clean and format the data. This is the final dataframe:

date	open	high	low	close	volume	dcl	dcm	dcu
2023-10-23 16:00:00-04:00	170.91	174.010	165.67	168.22	286078100	137.90	168.065	198.23
2023-10-30 16:00:00-04:00	169.02	177.780	167.90	176.65	310075880	141.32	169.775	198.23
2023-11-06 15:00:00-05:00	176.38	186.565	176.21	186.40	303653020	143.90	171.065	198.23
2023-11-13 15:00:00-05:00	185.82	190.960	184.21	189.69	262880720	143.90	171.065	198.23
2023-11-20 15:00:00-05:00	189.89	192.930	189.25	189.97	148351450	143.90	171.065	198.23

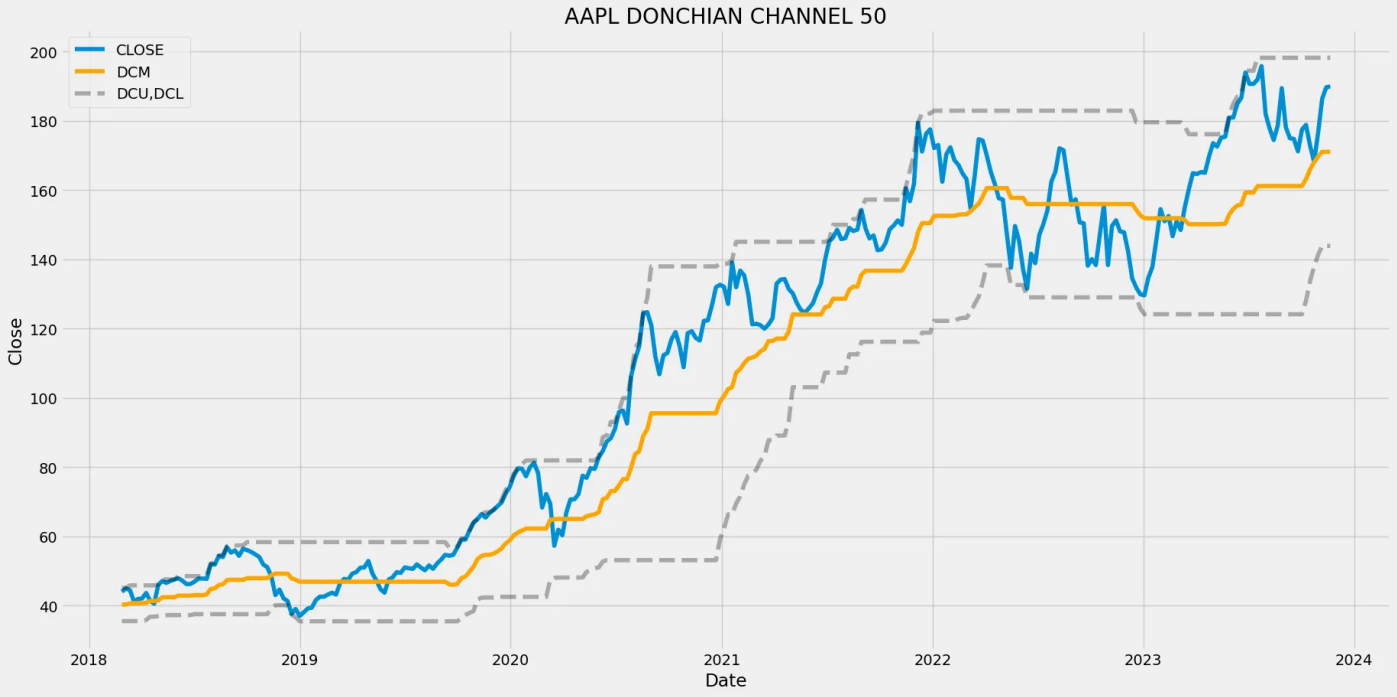
Donchian Channel of Apple stock (Image by Author)

To get a better view of the Donchian Channel indicator, let's plot the calculated values using the Matplotlib library:

```
# PLOTTING DONCHIAN CHANNEL

plt.plot(aapl[-300:].close, label = 'CLOSE')
plt.plot(aapl[-300:].dcl, color = 'black', linestyle = '--', alpha = 0.3)
plt.plot(aapl[-300:].dcm, color = 'orange', label = 'DCM')
plt.plot(aapl[-300:].dcu, color = 'black', linestyle = '--', alpha = 0.3, lal
plt.legend()
plt.title('AAPL DONCHIAN CHANNELS 50')
plt.xlabel('Date')
plt.ylabel('Close')
```

There is nothing much going on with this code. We are utilizing all the essential functions provided by `matplotlib` to create the visualization and this is the final chart:



Apple Donchian Channel Plot (Image by Author)

From the plot it can be observed that there are three important components in the Donchian Channel Indicator:

1. Upper Band: The upper band reveals the highest high of the stock over a specified period of time.
2. Lower Band: Basically the opposite of the upper band, it shows the lowest low of the stock over a specified period of time.
3. Middle Band: This component is a little different. It shows the average between the upper band and the lower band.

Donchian Channel is one of the most widely used indicators for observing breakouts happening in stock price movements and that's one of the core reasons for using it in this article.

Backtesting the Strategy

We have arrived at one of the most important steps in this article which is backtesting our breakout strategy. We are going to follow a very basic and straightforward system of backtesting for the sake of simplicity. The following code backtests the strategy and reveals its results:

```
# BACKTESTING THE STRATEGY

def implement_strategy(aapl, investment):

    in_position = False
    equity = investment

    for i in range(3, len(aapl)):
        if aapl['high'][i] == aapl['dcu'][i] and in_position == False:
            no_of_shares = math.floor(equity/aapl.close[i])
            equity -= (no_of_shares * aapl.close[i])
            in_position = True
            print(cl('BUY: ', color = 'green', attrs = ['bold']), f'{no_of_sl
        elif aapl['low'][i] == aapl['dcl'][i] and in_position == True:
            equity += (no_of_shares * aapl.close[i])
            in_position = False
            print(cl('SELL: ', color = 'red', attrs = ['bold']), f'{no_of_sha
    if in_position == True:
        equity += (no_of_shares * aapl.close[i])
        print(cl(f'\nClosing position at {aapl.close[i]} on {str(aapl.index[
        in_position = False

    earning = round(equity - investment, 2)
    roi = round(earning / investment * 100, 2)
    print(cl(f'EARNING: ${earning} ; ROI: {roi}%', attrs = ['bold']))

implement_strategy(aapl, 100000)
```

I’m not going to dive deep into the dynamics of this code as it will take some time to explain it. Basically, the program executes the trades based on the conditions that are satisfied. It enters the market when our entry condition is satisfied and exits when the exit condition is satisfied. These are trades executed by our program followed by the backtesting results:

```
BUY:  272340 Shares are bought at $0.3671875 on 1994-10-10
SELL:  272340 Shares are bought at $0.3214286 on 1995-10-09
BUY:  365659 Shares are bought at $0.2393975 on 1997-08-04
SELL:  365659 Shares are bought at $0.14508928 on 1997-12-29
BUY:  212688 Shares are bought at $0.2494422 on 1998-04-13
SELL:  212688 Shares are bought at $0.4598215 on 2000-09-25
BUY:  219244 Shares are bought at $0.4460715 on 2002-04-15
SELL:  219244 Shares are bought at $0.2671429 on 2002-07-15
BUY:  170827 Shares are bought at $0.3428571 on 2003-06-16
SELL:  170827 Shares are bought at $3.466786 on 2008-09-29
BUY:  97991 Shares are bought at $6.043571 on 2009-08-17
SELL:  97991 Shares are bought at $18.206929 on 2012-12-10
BUY:  89836 Shares are bought at $19.859643 on 2013-11-25
SELL:  89836 Shares are bought at $28.3225 on 2015-08-24
BUY:  85496 Shares are bought at $29.76 on 2017-01-09
SELL:  85496 Shares are bought at $37.6825 on 2018-12-17
BUY:  56768 Shares are bought at $56.7525 on 2019-09-30
SELL:  56768 Shares are bought at $137.59 on 2022-05-16
BUY:  44591 Shares are bought at $175.16 on 2023-05-15

Closing position at 189.97 on 2023-11-20
EARNING: $8371107.71 ; ROI: 8371.11%
```

AAPL Backtesting Results (Image by Author)

As I've claimed in the title of the article, our strategy has made an ROI of 8371% which is humongous. But it's time to see if our strategy has really outperformed the market.

SPY ETF Comparison

Drawing a comparison between the backtesting results of our strategy and the buy/hold returns of the SPY ETF helps us get a true sense of our strategy's performance. The following code calculates the returns of SPY ETF over the years:

```
spy = get_historical_data('SPY', '1993-01-01', '1W')
spy_ret = round(((spy.close.iloc[-1] - spy.close.iloc[0])/spy.close.iloc[0])

print(cl('SPY ETF buy/hold return:', attrs = ['bold']), f'{spy_ret}%')
```

In the above code, we are first extracting the historical data of SPY with the same specifications we used for AAPL. Then we are calculating the returns percentage of the index using a simple formula and this is the result:

```
SPY ETF buy/hold return: 936%
```

SPY ETF buy/hold return (Image by Author)

The return of the index is 936% which is actually pretty good but when compared to that of our strategy, there is a vast difference. Our strategy has outperformed the benchmark substantially and that's great news!

Closing Notes

In this article, we went through an extensive process of coding to backtest a simple yet very effective breakout strategy. And as expected, the results of the strategy were amazing. We started off with extracting the historical data of Apple using Benzinga's API, then slowly explored the Donchian Channel, and finally proceeded to backtest the strategy and compare the results with SPY ETF.

There are still a lot of aspects that can be improved. The backtesting system can be even more complex and realistic with the addition of brokerage commission and slippage. A proper risk management system must be in place, especially in the case of algo trading. Like these, there are many aspects to be improved which I'm leaving to you guys to explore.

With that being said, you've reached the end of the article. Before ending the article, I would like to give a shoutout to Benzinga for creating such a great library of APIs which includes institutional-grade market news & data APIs and I would suggest you guys check it out too. Hope you learned something new and useful today. Thank you for your time.

Finance Programming Python Data Science Technology

Follow

Written by Nikhil Adithyan

9.97K Followers · Writer for Level Up Coding

Founder @BacktestZone (https://www.backtestzone.com/), a no-code backtesting platform | Top Writer | Connect with me on LinkedIn: https://bit.ly/3yNuwCJ

More from Nikhil Adithyan and Level Up Coding

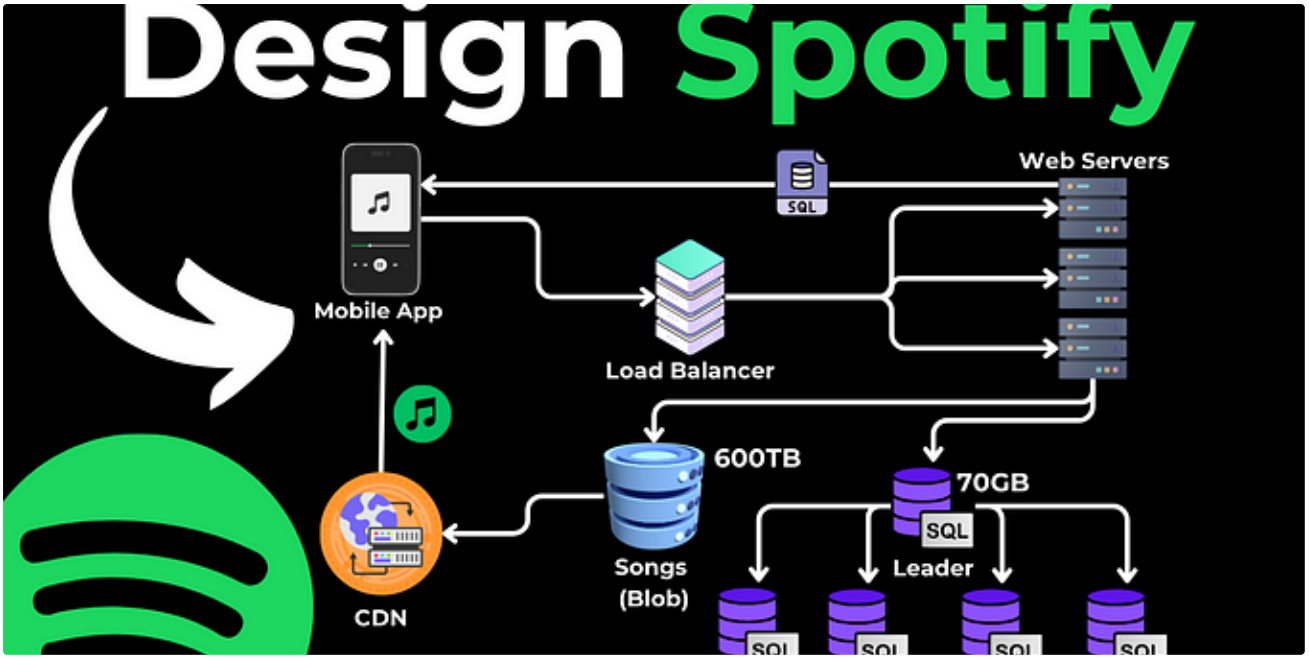
Nikhil Adithyan in Level Up Coding

Stock Market Sentiment Prediction with OpenAI and Python

An interesting exploration of the power of LLMs in stock analysis

11 min read · Feb 5, 2024

838 4



Hayk Simonyan in Level Up Coding

System Design Interview Question: Design Spotify

High-level overview of a System Design Interview Question - Design Spotify.

6 min read · Feb 21, 2024

3.3K 27

Tirendaz AI in Level Up Coding

How to Use ChatGPT in Daily Life?

Save time and money using ChatGPT

9 min read · Apr 4, 2023

4.3K 81

Nikhil Adithyan in Level Up Coding

Backtesting the Most Underrated SMA Trading Strategy which Beats the Market

And it's definitely not the SMA 50 & 200 crossover

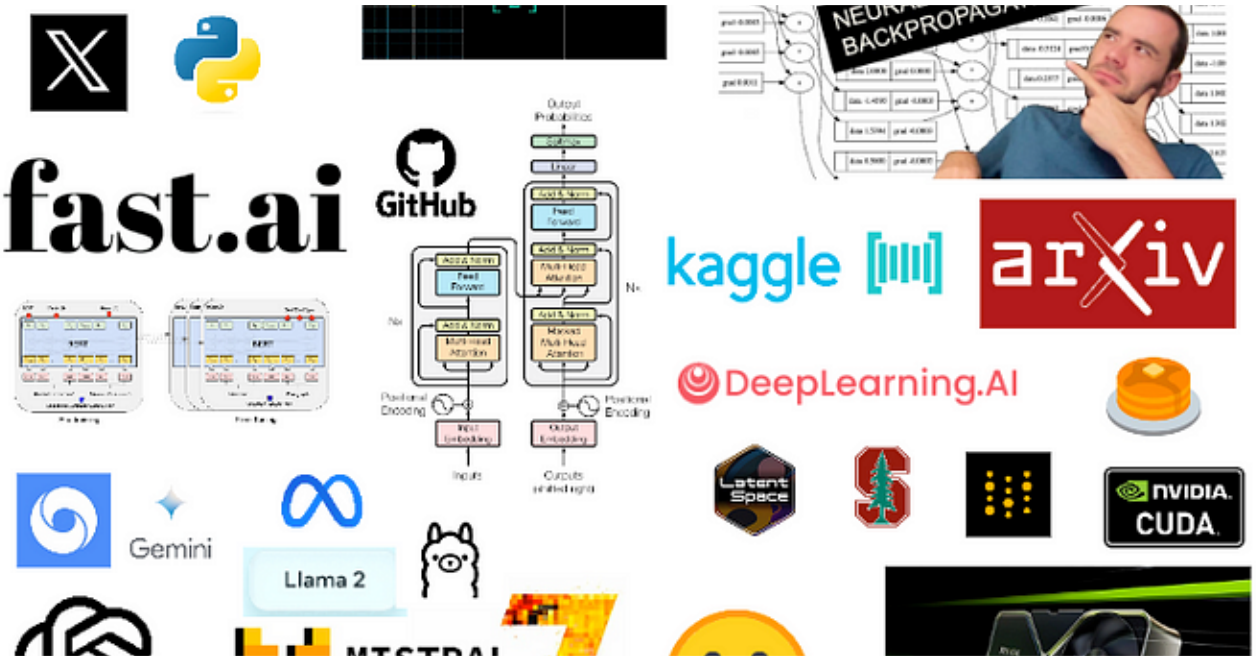
11 min read · Feb 28, 2024

434 6

See all from Nikhil Adithyan

See all from Level Up Coding

Recommended from Medium



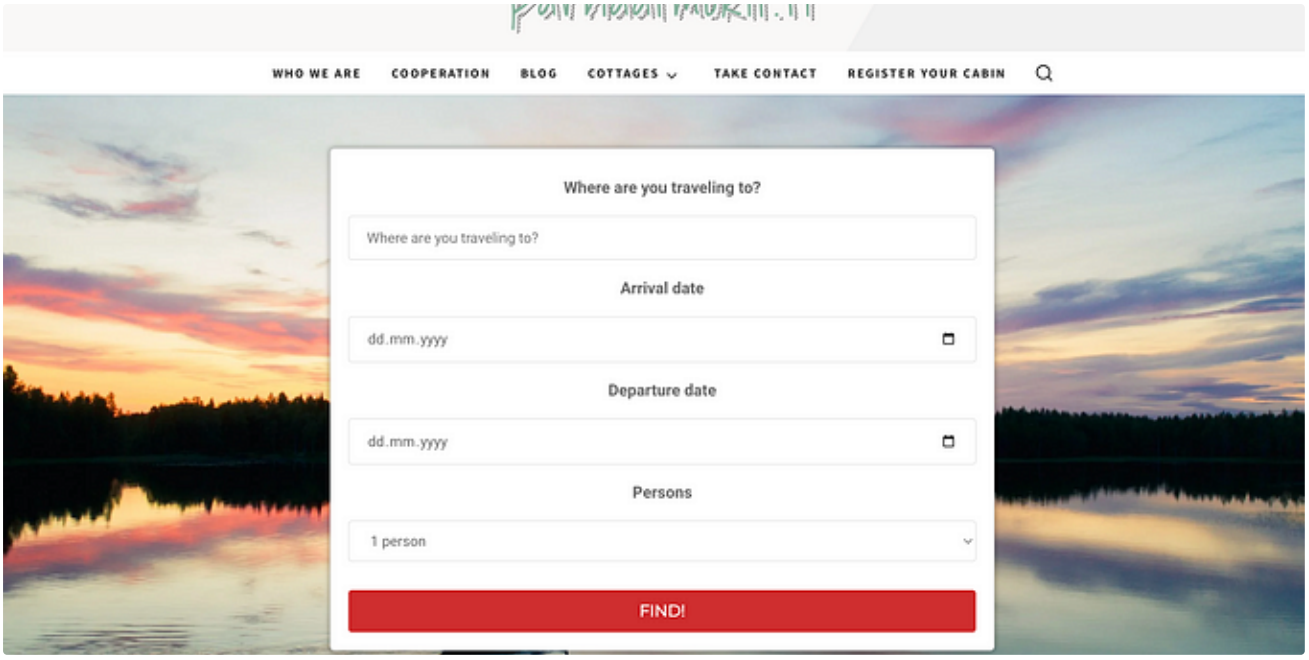
Benedict Neo in bitgrit Data Science Publication

Roadmap to Learn AI in 2024

A free curriculum for hackers and programmers to learn AI

11 min read · 8 hours ago

5.2K 62



Artturi Jalli

I Built an App in 6 Hours that Makes $1,500/Mo

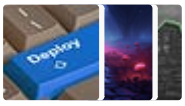
Copy my strategy!

✨ · 3 min read · Jan 23, 2024

13K 154

Lists

Coding & Development
11 stories · 494 saves



Predictive Modeling w/ Python
20 stories · 988 saves

ChatGPT prompts
46 stories · 1238 saves

General Coding Knowledge
20 stories · 999 saves

Vaishnav Manoj in DataX Journal

JSON is incredibly slow: Here's What's Faster!

Unlocking the Need for Speed: Optimizing JSON Performance for Lightning-Fast Apps and Finding Alternatives to it!

16 min read · Sep 28, 2023

14K 164

Mil Hoornaert in Long. Sweet. Valuable.

Stop Listening to Music, It Will Change Your Life

I stopped listening to music, find out what the results and benefits are in this post!

· 5 min read · Nov 23, 2023

8.3K 331

Paul Phoenix

6 Legit Apps To Make Truly Passive Income By Having Your Computer Turned On.

Discover how to earn passive income by simply leaving your computer running. Here are six methods that can help you monetize your idle...

9 min read · Jan 20, 2024

5.1K 93

Austin Starks in DataDrivenInvestor

My ChatGPT-Generated Trading Strategies are DEMOLISHING the Market.

No exaggeration. AI-Generated Portfolios are vastly superior

7 min read · Nov 16, 2023