Member-only story

Day 4 of 30 days of Data Engineering Series

Techniques for optimization...

Naina Chaturvedi · Follow

Published in Coders Mojo · 11 min read · Sep 5, 2022

Pic credits : sandbox

Welcome back peeps to Day 4 of Data Engineering!

What's covered in 30 days of Data Engineering with projects Series till now —

> *Day 1 : What's Data Engineering, Why Data Engineering, Data Engineers — ML Engineers — Data Scientists, Purpose and Scope*
>
> *Day 2 : Complete Python for Data Engineering — Part 1*

Day 3 : Complete Advanced Python for Data Engineering — Part 2

Day 4: Techniques to write efficient and Optimized Code

Day 5 : SQL

Day 6 : Advanced SQL

Day 7 : BigQuery and SQL vs NOSQL databases

Day 8 : Advanced Functions

Day 9 : Query Optimizations

Day 10 : MySQL and PostgreSQL

Day 11: Shell scripting and Linux “touch” command

Day 12 : Map Reduce, Data Warehouse, Data Lakes

Day 13: Pandas, Pandas, Data Cleaning and processing, Outlier Detection, Noisy Data, Missing Data, Pandas Functions, Aggregate Functions, Joins

Day 14 : Numpy

Day 15 : Advanced Pandas Techniques

Day 16 : Data Pre-processing, Handling missing values, Data Cleaning, Mean/mode/median Imputation, Hot Deck Imputation, Rescale Data, Binarize Data, Regression Imputation, Stochastic regression imputation, Feature Scaling

Day 17 : Data Augmentation, Read and Process Large Datasets

Day 18 : Data Visualization basics, Data Visualization Projects, Data Visualization using Plotly and Bokeh, Data Profiling, Summary Functions, Indexing, Grouping, Linear Regression, Multi Linear Regression, Polynomial Regression, Regression, Support Vector Regression, Decision Tree Regression, Random Forest Regression, Feature Engineering, GroupBy Features, Categorical and Numerical Features, Missing Value Analysis, Fill the missing Values, Unique Value Analysis, Univariate Analysis, Bivariate Analysis, Multivariate Analysis, Correlation Analysis, Spearman's ρ, Pearson's r, Kendall's τ, Cramér's V (φc), Phik (φk)

Day 19 : MySQL and PostgreSQL

. . .

System Design Case Studies — In Depth

Design Instagram

Design Messenger App

Design Twitter

Design URL Shortener

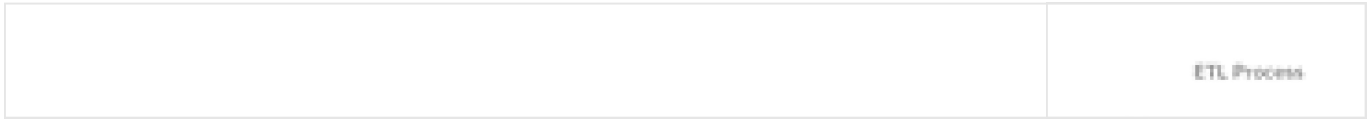
Design Dropbox

Mega Compilation : Solved System Design Case studies

. . .

Pre-requisite to Day 4 is to complete Day 1–3(link below):

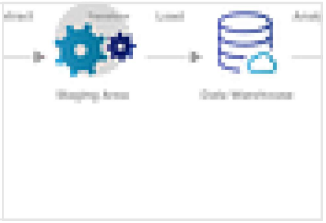
Day 1 of 30 days of Data Engineering can be found below —



Day 1 of 30 days of Data Engineering

With examples and projects...

medium.com

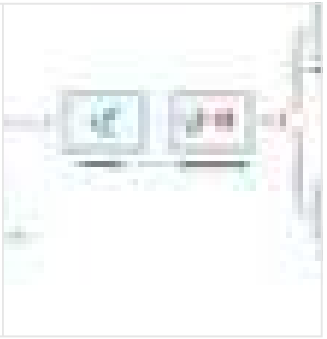


Day 2 of 30 days of Data Engineering can be found below —

Day 2 of 30 days of Data Engineering

With examples and projects...

medium.com

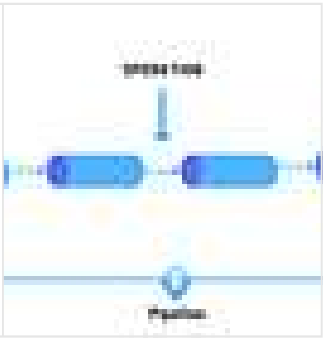


Day 3 of 30 days of Data Engineering can be found below —

Day 3 of 30 days of Data Engineering Series

With examples and projects ...

medium.com



. . .

Projects Videos —

All the projects, data structures, SQL, algorithms, system design, Data Science and ML , Data Analytics, Data Engineering, , Implemented Data Science and ML projects, Implemented Data Engineering Projects, Implemented Deep Learning Projects, Implemented Machine Learning Ops Projects, Implemented Time Series Analysis and Forecasting Projects, Implemented Applied Machine Learning Projects, Implemented Tensorflow and Keras Projects, Implemented PyTorch Projects, Implemented Scikit Learn Projects, Implemented Big Data Projects, Implemented Cloud Machine Learning Projects, Implemented Neural Networks Projects, Implemented OpenCV Projects,Complete ML Research Papers Summarized, Implemented Data Analytics projects, Implemented Data Visualization Projects, Implemented Data Mining Projects, Implemented Natural Leaning Processing Projects, MLOps and Deep Learning, Applied Machine Learning with Projects Series, PyTorch with Projects Series, Tensorflow and Keras with Projects Series, Scikit Learn Series with Projects, Time Series Analysis and Forecasting with Projects Series, ML System Design Case Studies Series videos will be published on our youtube channel (just launched).

Subscribe today!

. . .

This is Day 4 of 30 days of Data Engineering Series where we will be covering —

Techniques to write efficient and Optimized Code

. . .

Our whole syllabi for 30 days of Data Engineering —

I'l be covering only the most important topics in Data Engineering **with projects** (written below) —

1. Data Engineering

> *What's Data Engineering*
>
> *Why Data Engineering*
>
> *Data Engineers — ML Engineers — Data Scientists*
>
> *Purpose and Scope*

2. Python for Data Engineering

> *Basic Python with Project*
>
> *Advanced Python with Project*
>
> *Techniques to write efficient and optimized code*

3. Scripting and Automation

> *Shell Scripting*

> *CRON*
>
> *ETL*

4. Relational Databases and SQL

> *RDBMS*
>
> *Data Modeling*
>
> *Basic SQL*
>
> *Advanced SQL*
>
> ***Big Query***

5. NoSQL Data bases and Map Reduce

> *Unstructured Data*
>
> *Advanced ETL*

> *Map-Reduce*
>
> *Data Warehouses*
>
> *Data API*

6.Data Analysis

> *Pandas*
>
> *Numpy*
>
> *Web Scraping*
>
> *Data Visualization*

7. Data Processing Techniques

> *Batch Processing : Apache Spark*
>
> *Stream Processing — Spart Streaming*

> *Build Data Pipelines*
>
> *Target Databases*
>
> *Machine learning Algorithms*

8. Big Data

> *Big data basics*
>
> *HDFS in detail*
>
> *Hadoop Yarn*
>
> *Sqoop Hadoop*
>
> *Hadoop Yarn*
>
> *Hive*
>
> *Pig*

> *Hbase*

9. WorkFlows

> *Introduction to Airflow*
>
> *Airflow hands on project*

10. Infrastructure

> *Docker*
>
> *Kubernetes*
>
> *Business Intelligence*

11. Cloud Computing

> *AWS*
>
> *Google Cloud Platform*

12. Research Papers — Data Engineering

> *Some amazing research papers- data engineering that I have read over the years to help you boot up to the industry standards and what's next in this field.*

. . .

Lets dive in!

Some of the most important optimization techniques are -

1. ***Use built-in functions and libraries:*** *Python has a lot of built-in functions and libraries that are optimized for performance. Using them can save a lot of time and memory.*
2. ***Avoid using global variables:*** *Global variables can slow down the performance of your code and make it harder to debug.*
3. ***Use list comprehensions:*** *List comprehensions are a more efficient way to create and manipulate lists in Python.*

4. ***Use generators:*** *Generators are a way to create iterators in Python. They are more memory-efficient than lists because they only generate values on-the-fly as they are needed.*
5. ***Use the "join" method instead of "+" for strings:*** *The "+" operator creates a new string each time it is used, which can slow down your code. The "join" method is faster and more memory-efficient.*
6. ***Use "in" operator instead of "index" method for lists:*** *The "in" operator is faster for checking if an element is in a list.*
7. ***Avoid using unnecessary loops:*** *Unnecessary loops can slow down your code and use up more memory.*
8. ***Use the "multiprocessing" module for parallel processing:*** *The "multiprocessing" module allows you to run multiple processes in parallel, which can speed up your code.*
9. ***Use "numpy" for numerical computations:*** *The numpy library is highly optimized for numerical computations and can be significantly faster than pure Python code.*
10. ***Profile and Optimize:*** *Use profilers like cProfile, line_profiler, memory_profiler, etc. to profile and optimize your code.*

Complete Code —

```
import time
import string
import random
from multiprocessing import Pool
import numpy as np
import cProfile

# Use built-in functions and libraries
result = sum([i for i in range(1000)])  # Using list comprehension and sum()

# Avoid using global variables
def calculate_sum(numbers):
    return sum(numbers)

numbers = [1, 2, 3, 4, 5]
sum_result = calculate_sum(numbers)

# Use list comprehensions
squares = [x**2 for x in range(10)]

# Use generators
def random_numbers(n):
    for _ in range(n):
        yield random.randint(1, 100)

for num in random_numbers(5):
    print(num)

# Use the "join" method instead of "+"
letters = string.ascii_lowercase
```

```
joined_string = ''.join(letters)

# Use "in" operator instead of "index" method for lists
my_list = [1, 2, 3, 4, 5]
if 3 in my_list:
    print("Element found!")

# Avoid using unnecessary loops
data = [1, 2, 3, 4, 5]
filtered_data = [x for x in data if x > 2]

# Use the "multiprocessing" module for parallel processing
def square_number(n):
    return n**2

if __name__ == '__main__':
    numbers = [1, 2, 3, 4, 5]
    with Pool(processes=2) as pool:
        squared_numbers = pool.map(square_number, numbers)

# Use "numpy" for numerical computations
arr = np.array([1, 2, 3, 4, 5])
mean_value = np.mean(arr)

# Profile and Optimize
def perform_task():
    time.sleep(1)

profiled_task = cProfile.Profile()
profiled_task.enable()
perform_task()
profiled_task.disable()
profiled_task.print_stats()

import time
```

```
# Technique 1: Use appropriate data structures
# Use sets for membership tests and to eliminate duplicates
my_list = [1, 2, 3, 4, 5, 1, 2, 3]
my_set = set(my_list)
print(my_set)  # Output: {1, 2, 3, 4, 5}

# Technique 2: Avoid unnecessary computations or evaluations
# Use short-circuiting for logical operators
x = 5
y = 10
if x > 0 and y > 5:
    print("Condition satisfied")

# Technique 3: Optimize loops and iterations
# Use list comprehension instead of traditional for loops
numbers = [1, 2, 3, 4, 5]
squared_numbers = [num ** 2 for num in numbers if num % 2 == 0]
print(squared_numbers)  # Output: [4, 16]

# Technique 4: Minimize function calls or method invocations
# Store frequently used values in variables
def expensive_calculation(num):
    # Expensive calculation
    time.sleep(1)
    return num ** 2

result = expensive_calculation(5)
print(result)  # Output: 25

# Technique 5: Utilize built-in functions and libraries
# Use built-in functions for common operations
my_list = [1, 2, 3, 4, 5]
total = sum(my_list)
print(total)  # Output: 15

# Technique 6: Profile and optimize critical sections
```

```
# Identify bottlenecks and optimize accordingly
start_time = time.time()

# Critical section of code
time.sleep(2)

end_time = time.time()
execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")
```

. . .

In python, Enumerate is used to write efficient python code. Many a times we need to keep a count of iterations. Python's enumerate takes a collection i.e iterable, adds counter to it and returns it as an enumerate object

Syntax :

enumerate(iterable, start=0)

Implementation —

```
"""
Enumerate : Use enumerate() function : Python's enumerate takes a
collection i.e iterable, adds counter to it and returns it as an
enumerate object.
"""

countries = ['USA','Canada','Singapore','Taiwan']
enum_countries = enumerate(countries)

enumerate_countries = enumerate(countries,5)
print(list(enumerate_countries))
print(type(enumerate_countries))
```

Output —

```
[(5, 'USA'), (6, 'Canada'), (7, 'Singapore'), (8, 'Taiwan')]
<class 'enumerate'>
```

Implementation 2 —

```
countries = ['USA','Canada','Singapore','Taiwan']
for i,item in enumerate(countries):
    print(i,item)
```

Output —

```
0 USA
1 Canada
2 Singapore
3 Taiwan
```

. . .

Some of the other best Series —

> *30 days of Machine Learning Ops*
>
> *How to solve any System Design Question (approach that you can take)?*
>
> *Complete System Design Case Studies Series*
>
> *30 Days of Natural Language Processing (NLP) Series*
>
> *30 days of Data Structures and Algorithms and System Design Simplified*

60 Days of Deep Learning with Projects Series

60 Days of Deep Learning with Projects Series

30 days of Data Engineering with projects Series

Data Science and Machine Learning Research (papers) Simplified **

60 days of Data Science and ML Series with projects

100 days : Your Data Science and Machine Learning Degree Series with projects

23 Data Science Techniques You Should Know

Tech Interview Series — Curated List of coding questions

Complete System Design with most popular Questions Series

Complete Data Visualization and Pre-processing Series with projects

Complete Python Series with Projects

> *Complete Advanced Python Series with Projects*
>
> *Kaggle Best Notebooks that will teach you the most*
>
> *Complete Developers Guide to Git*
>
> *Exceptional Github Repos — Part 1*
>
> *Exceptional Github Repos — Part 2*
>
> *All the Data Science and Machine Learning Resources*
>
> *210 Machine Learning Projects*

Tech Newsletter —

> *If you are interested, you can join my newsletter through which I send tech interview tips, techniques, patterns, hacks — Software Development, ML, Data Science, Startups and Technology projects to more than 30K readers. You can subscribe to* ***Tech Brew :***

Ignito

Data Science, ML, AI and more... Click to read Ignito, by Naina Chaturvedi, a Substack publication. Launched 7 months...

naina0405.substack.com

Github —

Coder-World04 - Overview

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

. . .

In python, Zip takes one or more iterables(list,tuples etc) and aggregates them into tuple and returns the iterator object

Syntax :

*zip(*iterators)*

Implementation —

```
# Use Zip : Zip takes one or more iterables and aggregates them into
# tuple and returns the iterator object

name = ["Steve","Paul","Brad"]
roll_no = [4,1,3]
marks = [20,40,50]

mapped = zip(name,roll_no,marks)
mapped = set(mapped)
print(mapped)
```

Output —

```
{('Brad', 3, 50), ('Steve', 4, 20), ('Paul', 1, 40)}
```

. . .

To make code work faster use builtin functions and libraries like map() which applies a function to every member of iterable sequence and returns the result.

Implementation —

```
"""
Map function : In Python, map() function applies the given function
#to each item of a given iterable construct (i.e lists, tuples etc)
and returns a map object.
"""

numbers =(100,200,300)
result = map(lambda x:x+x,numbers)
total = list(result)
print(total)
```

Output —

```
[200, 400, 600]
```

. . .

NumPy arrays are homogeneous and provide a fast and memory efficient alternative to Python lists.NumPy arrays vectorization technique, vectorize operations so they are performed on all elements of an object at once which

allows the programmer to efficiently perform calculations over entire arrays.

Implementation —

```
import numpy as np
def reciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0/values[i]
    return output

values  = np.random.randint(1,15,size=6)
reciprocals(values)
```

Output —

```
array([0.25      , 0.5       , 0.1       , 0.16666667, 0.14285714,
       0.07142857])
```

. . .

To swap the variables, use multiple assignment

Implementation —

```
# Use multiple assignment

f_name,l_name,city = "Steve","Paul","NewYork"

print(f_name,l_name,city)

#To swap variable

a = 5
b = 10

a,b = b,a
print(a,b)
```

Output —

```
Steve Paul NewYork
10 5
```

. . .

Use Comprehensions

Implementation —

```
#List Comprehension

list_two = [5,10,15,20,20,40,50,60]
new_list = [x**3 for x in list_two]
print(new_list)

#Dictionary Comprehension

dict_one = [1,2,3,4]
new_dict = {x:x**2 for x in dict_one if x%2 ==0}
print(new_dict)
```

Output —

```
[125, 1000, 3375, 8000, 8000, 64000, 125000, 216000]
{2: 4, 4: 16}
```

. . .

Membership : To check if membership of a list, it's generally faster to use the "in" keyword

Implementation —

```
days = ["sunday","monday","tuesday"]
for d in days:
    print('Today is {}'.format(d))
print('tuesday' in days)
print('friday' in days)
```

Output —

```
Today is sunday
Today is monday
Today is tuesday
True
False
```

. . .

Counter : Counter is one of the high performance container data types

Implementation —

```
from collections import Counter
sample_dict = {'a':4,'b':8,'c':2}
print(Counter(sample_dict))
```

Output —

```
Counter({'b': 8, 'a': 4, 'c': 2})
```

. . .

Python Itertools are fast, memory efficient functions — a collection of constructs for handling iterators.

Implementation —

```
import itertools
for i in itertools.count(30,4):
    print(i)
    if i>30:
        break
```

Output —

```
30
34
```

Implementation 2 —

```
import itertools
countries =[("West","USA"), ("East","Singapore"),("West","Canada"),
("East","Taiwan")]

iter_one = itertools.groupby(countries,lambda x:x[0])
for key,group in iter_one:
    result = {key:list(group)}
    print(result)
```

Output —

```
{'West': [('West', 'USA')]}
{'East': [('East', 'Singapore')]}
{'West': [('West', 'Canada')]}
{'East': [('East', 'Taiwan')]}
```

. . .

Use sets to remove duplicates

Implementation —

```
s1 = {1,2,4,6,0,3,2,1,7,4,3}
s1.add(10)
s1.update([12,13])
print(s1)
```

Output —

```
{0, 1, 2, 3, 4, 6, 7, 10, 12, 13}
```

. . .

Use Generators

Range (range()) uses lazy evaluation, so instead of range() use xrange() which returns the generator object

Implementation —

```
def test_sequence():
    num = 0
    while num<10:
        yield num
        num+=1

for i in test_sequence():
    print(i,end=",")
```

Output —

```
0,1,2,3,4,5,6,7,8,9,
```

. . .

Practice writing idiomatic code as it will make your code run faster

. . .

Examine Runtime of your code snippet

Implementation —

```
%timeit ('x=3; L=[x**n for n in range(20)]')
```

Output —

```
12.9 ns ± 0.894 ns per loop (mean ± std. dev. of 7 runs, 10000000
```

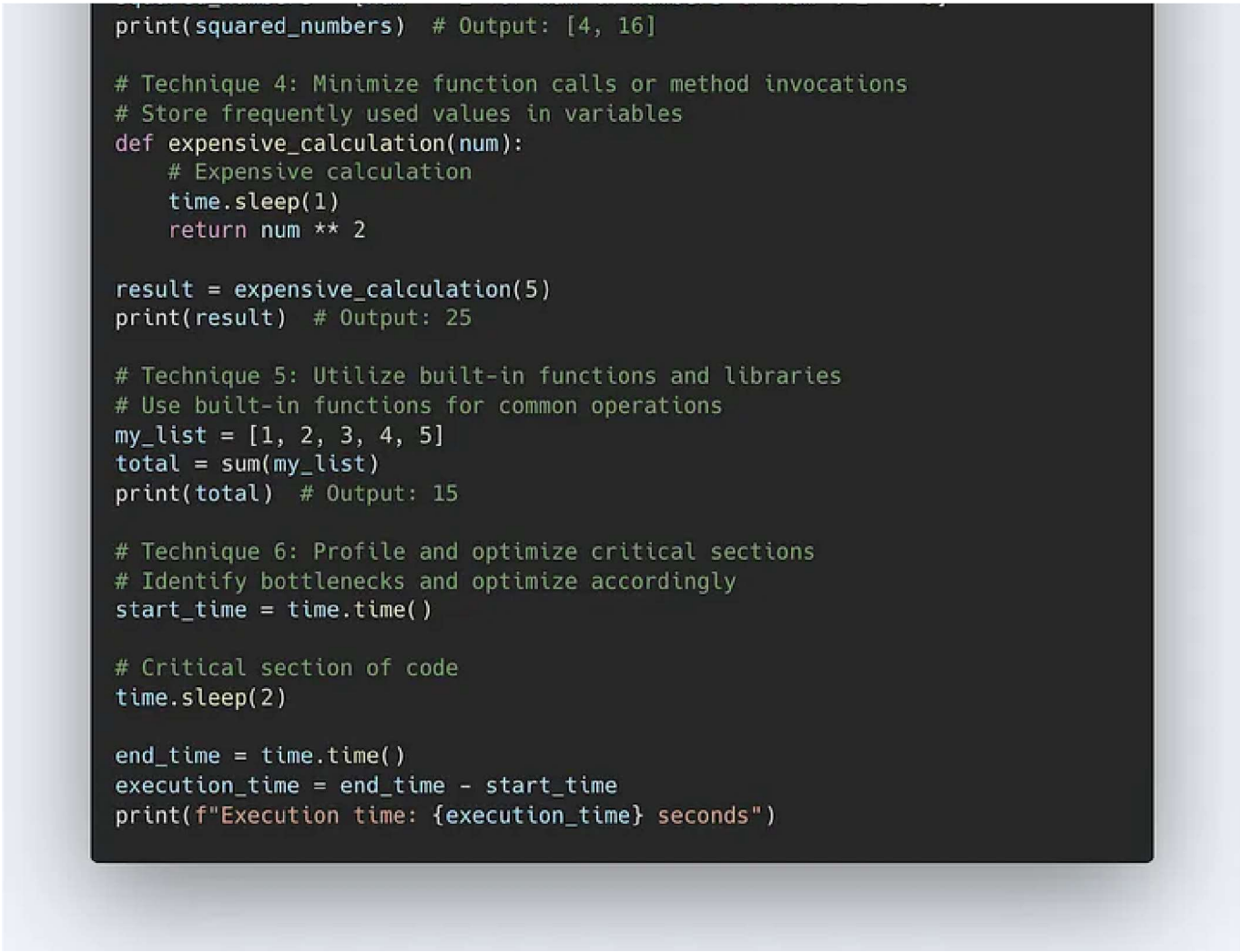
```
loops each)
```

. . .

Complete Python code —

```python
# Author : Naina Chaturvedi

import time
import string
import random
from multiprocessing import Pool
import numpy as np
import cProfile

# Use built-in functions and libraries
result = sum([i for i in range(1000)])  # Using list comprehension and sum()

# Avoid using global variables
def calculate_sum(numbers):
    return sum(numbers)

numbers = [1, 2, 3, 4, 5]
sum_result = calculate_sum(numbers)

# Use list comprehensions
squares = [x**2 for x in range(10)]

# Use generators
def random_numbers(n):
    for _ in range(n):
        yield random.randint(1, 100)

for num in random_numbers(5):
    print(num)

# Use the "join" method instead of "+"
letters = string.ascii_lowercase
joined_string = ''.join(letters)

# Use "in" operator instead of "index" method for lists
my_list = [1, 2, 3, 4, 5]
if 3 in my_list:
    print("Element found!")
```

```
# Avoid using unnecessary loops
data = [1, 2, 3, 4, 5]
filtered_data = [x for x in data if x > 2]

# Use the "multiprocessing" module for parallel processing
def square_number(n):
    return n**2

if __name__ == '__main__':
    numbers = [1, 2, 3, 4, 5]
    with Pool(processes=2) as pool:
        squared_numbers = pool.map(square_number, numbers)

# Use "numpy" for numerical computations
arr = np.array([1, 2, 3, 4, 5])
mean_value = np.mean(arr)

# Profile and Optimize
def perform_task():
    time.sleep(1)

profiled_task = cProfile.Profile()
profiled_task.enable()
perform_task()
profiled_task.disable()
profiled_task.print_stats()


# Technique 1: Use appropriate data structures
# Use sets for membership tests and to eliminate duplicates
my_list = [1, 2, 3, 4, 5, 1, 2, 3]
my_set = set(my_list)
print(my_set)  # Output: {1, 2, 3, 4, 5}

# Technique 2: Avoid unnecessary computations or evaluations
# Use short-circuiting for logical operators
x = 5
y = 10
if x > 0 and y > 5:
    print("Condition satisfied")

# Technique 3: Optimize loops and iterations
# Use list comprehension instead of traditional for loops
numbers = [1, 2, 3, 4, 5]
squared_numbers = [num ** 2 for num in numbers if num % 2 == 0]
```



```
print(squared_numbers)  # Output: [4, 16]

# Technique 4: Minimize function calls or method invocations
# Store frequently used values in variables
def expensive_calculation(num):
    # Expensive calculation
    time.sleep(1)
    return num ** 2

result = expensive_calculation(5)
print(result)  # Output: 25

# Technique 5: Utilize built-in functions and libraries
# Use built-in functions for common operations
my_list = [1, 2, 3, 4, 5]
total = sum(my_list)
print(total)  # Output: 15

# Technique 6: Profile and optimize critical sections
# Identify bottlenecks and optimize accordingly
start_time = time.time()

# Critical section of code
time.sleep(2)

end_time = time.time()
execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")
```

. . .

System Design Case Studies — In Depth

Design Instagram

Design Messenger App

. . .

All the Complete System Design Series Parts —

> 1. System design basics
>
> 2. Horizontal and vertical scaling
>
> 3. Load balancing and Message queues
>
> 4. High level design and low level design, Consistent Hashing, Monolithic and Microservices architecture
>
> 5. Caching, Indexing, Proxies
>
> 6. Networking, How Browsers work, Content Network Delivery (CDN)
>
> 7. Database Sharding, CAP Theorem, Database schema Design

> 8. Concurrency, API, Components + OOP + Abstraction
>
> 9. Estimation and Planning, Performance
>
> 10. Map Reduce, Patterns and Microservices
>
> 11. SQL vs NoSQL and Cloud
>
> 12. Most Popular System Design Questions

Github —

Complete-System-Design/README.md at main · Coder-World04/Complete-System-Design

This repository contains everything you need to become proficient in System Design Topics you should know in System...

github.com

. . .

Keep learning and coding ;)

Day 5 coming soon!

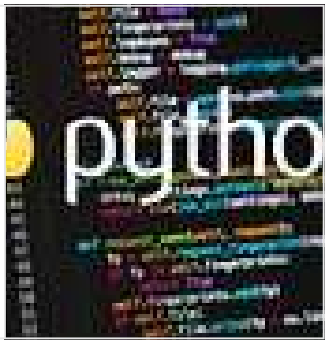
. . .

For Python Projects —

Complete Python And Projects — Mega Compilation

Everything that you need to know in Python with Projects...

medium.com



Analyzing Video using Python, OpenCV and NumPy

With Code Implementation...

medium.datadriveninvestor.com

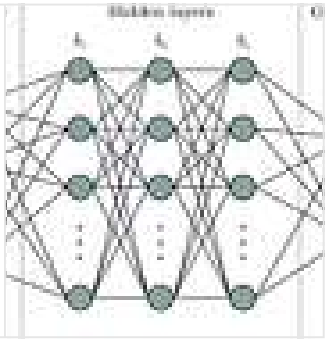
. . .

For complete 60 days of Data Science and ML : Day 1 — Day 60 : Quick Recap of 60 days of Data Science and ML

Day 1 — Day 60 : Quick Recap of 60 days of Data Science and ML

Connect the ML dots...

medium.com



Follow for more updates. Stay tuned and keep coding! *Disclosure: Some of the links are affiliates.*

. . .

For other projects, tune to —

Build Machine Learning Pipelines(With Code)

Build Machine Learning Pipelines(With Code) — Part 1

Complete implementation...

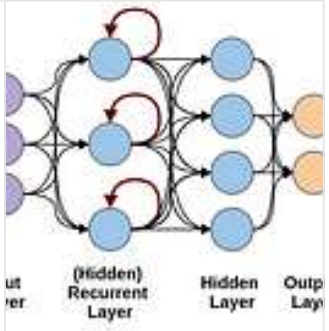
medium.datadriveninvestor.com

Recurrent Neural Network with Keras

Recurrent Neural Network with Keras

Project Implementation and cheatsheet...

medium.datadriveninvestor.com

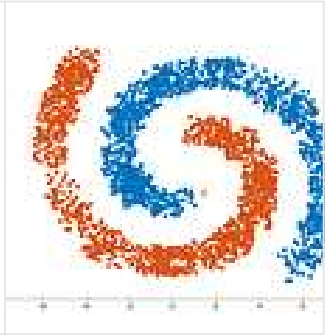


Clustering Geolocation Data in Python using DBSCAN and K-Means

Clustering Geolocation Data in Python using DBSCAN and K-Means

Project Implementation...

medium.datadriveninvestor.com



Facial Expression Recognition using Keras

Facial Expression Recognition using Keras

Project Implementation...

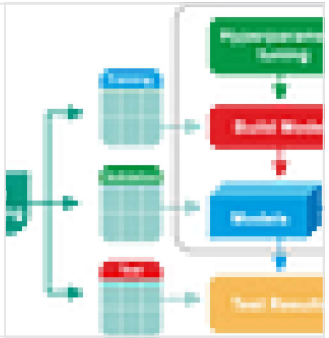
medium.datadriveninvestor.com

Hyperparameter Tuning with Keras Tuner

Hyperparameter Tuning with Keras Tuner

Project Implementation....

medium.datadriveninvestor.com

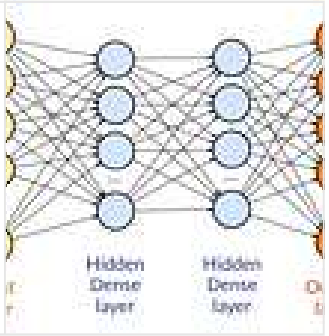


Custom Layers in Keras

Custom Layers in Keras

Code implementation ...

medium.datadriveninvestor.com



Machine Learning | Data Science | Artificial Intelligence | Programming | Tech

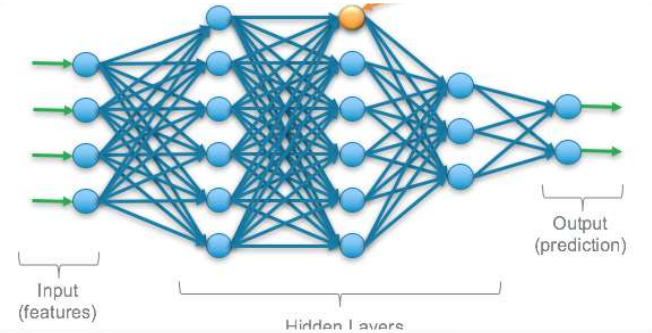
Written by Naina Chaturvedi

Follow

8.9K Followers · Editor for Coders Mojo

us,World Traveler, Sr. SDE, Researcher Cornell Uni, Women in Tech, Coursera Instructor ML & GCP, Trekker, IITB,Reader,I write for fun@AI & Python publications

More from Naina Chaturvedi and Coders Mojo



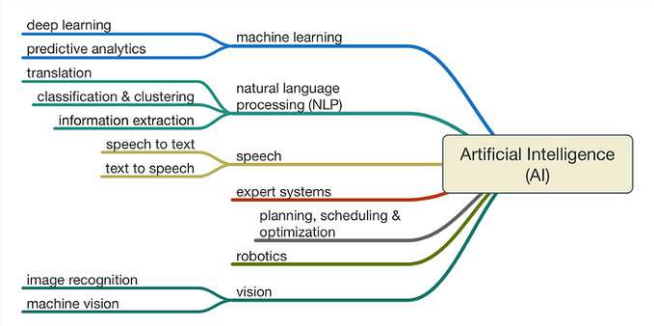
Naina Chaturvedi in Coders Mojo

Day 1 of 60 Days of Deep Learning with Projects Series

With Projects and Examples...

10 min read · Aug 19, 2022

397 3



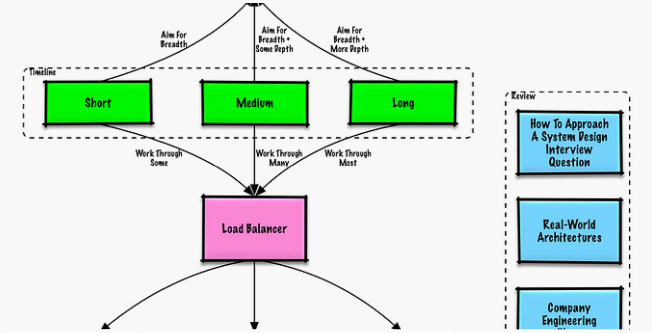
Naina Chaturvedi in Coders Mojo

Complete ML/DL Research Papers Summarized Series

Repo for all the Research Papers(vertical post)...

225 min read · Jan 10

115



Naina Chaturvedi in Coders Mojo

Naina Chaturvedi in Python in Plain English

Most Popular System Design Questions—Mega Compilation

Just for your reference...

12 min read · May 2, 2022

169

Python Iterators, Generators And Decorators Made Easy

A Quick Implementation Guide

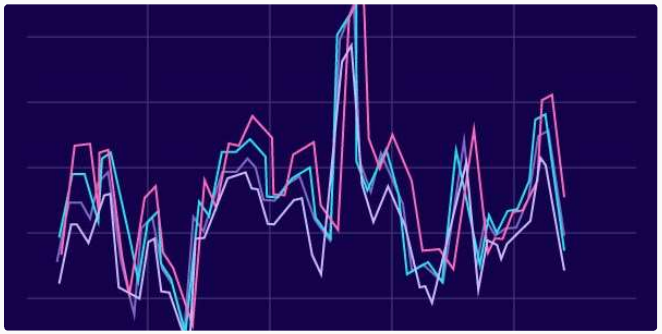
11 min read · Sep 16, 2020

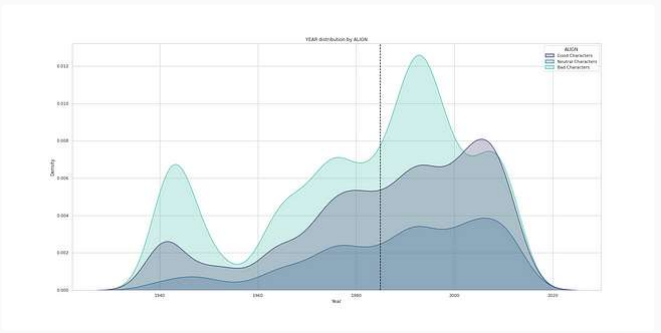
481 2

See all from Naina Chaturvedi

See all from Coders Mojo

Recommended from Medium





Naina Chaturvedi in Coders Mojo

Day 1 of 15 Days of Time Series Analysis and Forecasting with...

Welcome back peeps. Happy to share that we have just finished —

✦ · 6 min read · Dec 16, 2022

141

Naina Chaturvedi in Coders Mojo

Implemented Data Visualization Projects

Repo for all the projects (vertical post)...

✦ · 6 min read · Jan 10

113

Lists

What is ChatGPT?

9 stories · 103 saves

Stories to Help You Grow as a Software Developer

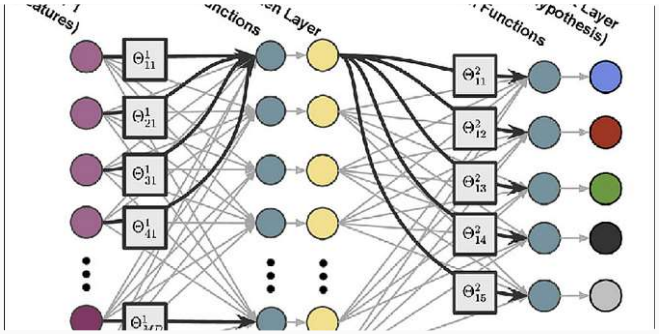
19 stories · 123 saves

Leadership

30 stories · 58 saves

Stories to Help You Level-Up at Work

19 stories · 98 saves



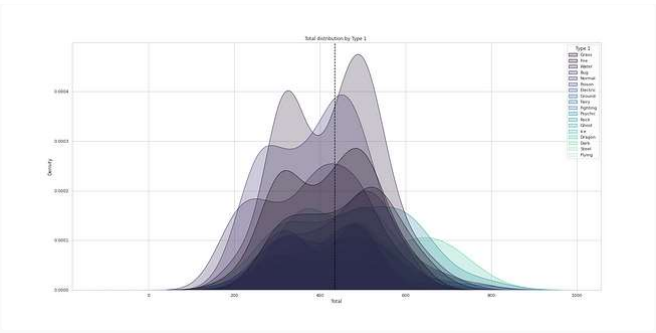
Naina Chaturvedi in Coders Mojo

Implemented Deep Learning Projects

Repo for all the projects (vertical post)...

✦ · 99 min read · Jan 6

116



Naina Chaturvedi in Coders Mojo

Day 16 of 30 days of Data Engineering Series with Projects

Welcome back peeps to Day 16 of Data Engineering Series with Projects!

✦ · 13 min read · Dec 24, 2022

103

Youssef Hosni in Level Up Coding

Naina Chaturvedi in Coders Mojo

13 SQL Statements for 90% of Your Data Science Tasks

Structured Query Language (SQL) is a programming language designed for...

15 min read · Feb 27

2.5K 26

Day 11 of 30 days of Data Engineering Series with Projects

Welcome back peeps to Day 11 of Data Engineering Series with Projects! In this we...

12 min read · Dec 23, 2022

101

See more recommendations