

Noor.AI – Skincare Product Recommendation Feature Development Report

1. Introduction

The Skincare Product Recommendation feature of Noor.AI helps users discover suitable skincare products based on their selected skin type. This standalone module was built using Python on Google Colab and was developed separately for testing before full integration into the Noor.AI web application. The recommendation logic is based on a Kaggle dataset of skincare products, which is filtered according to the user's selected skin type (dry, oily, or normal). If a product is marked suitable for all three skin types, it is also included in the recommended results. Once the logic was tested in Python, the module was turned into a Flask server with help from Gemini, and a simple HTML frontend was created with assistance from Claude.ai.

2. Tools & Technologies Used

| Tool | Purpose |
|-------------------------|---|
| Python | Main language for building the recommendation system |
| Google Colab | Environment for developing and testing Python logic |
| Kaggle | Source of the skincare product dataset |
| Flask | Used to serve the recommendation feature as a web API |
| Git & GitHub | Version control and backup |
| Claude.ai | Helped build the HTML page for the frontend |
| Gemini | Helped integrate Python code into a Flask web server |

3. Dataset & Preprocessing

Used a skincare product dataset from Kaggle, which included product names, descriptions, ingredients, and associated skin types.

Data was loaded and processed in Python using Pandas for easier filtering.

Each product was categorized under one or more skin types: dry, oily, normal, or all.

4. Recommendation System Design

A simple rule-based filtering logic was implemented using Python:

- If the user selects dry skin, products marked for dry and all skin types are shown.
- The same logic applies for oily and normal skin types.
- The system was first tested in Google Colab to validate filtering accuracy.
- After confirming the logic, the Python code was adapted into a Flask server.
- The final step was to connect the Flask API to an HTML form, where users could select their skin type and receive product suggestions.

5. Feature Workflow

1. User selects a skin type (dry, oily, normal) from a dropdown menu on the frontend.
2. The selection is sent to the Flask backend via a form submission.
3. Backend filters the product dataset and returns a list of suitable products:
 - Products explicitly match the selected skin type.
 - Products marked as suitable for all skin types.
4. Results are displayed on the HTML page for the user to view.

6. Challenges Faced

- Preparing and cleaning the dataset to ensure each product was properly categorized.
- Handling the additional logic for multi-type or “all skin type” products.
- Figuring out how to connect the Python logic to a working Flask server.
- Testing the dynamic display of results on the frontend HTML page.
- Validating feature accuracy and ensuring a consistent user experience.

7. Future Improvements

- Add more skin concerns such as acne-prone, sensitive, or combination.
- Include filters like product price, brand, or key ingredients.
- Add product links, allowing users to access the official websites to purchase recommended skincare products.
- Enhance UI/UX by adding images, user reviews, and better layout design.

Created by Shafaq

8. Screenshots

Step 1: Dataset and Filtering Logic

Google Colab was used to test and refine filtering logic based on skin type, including support for products that are suitable for all.

```

Skincare_recommendations.py:nb
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text + Run all +
1 import pandas as pd
2 from IPython.display import display, HTML
3
4 # Load your dataset
5 df = pd.read_csv('cosmetic_p.csv') # Replace with actual filename
6
7 # Show first 10 rows (original dataset)
8 print('ORIGINAL DATASET (First 10 Rows):')
9 display(df.head(10).style.set_caption('Original Dataset Sample').set_table_styles(
10     [{"selector": "caption", "props": [{"caption_side": "top", "font_weight": "bold"}]}])
11
12
13 ORIGINAL DATASET (First 10 Rows):
14
15 Original Dataset Sample
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006

```

Step 2: Flask Integration

With Gemini's help, the logic was wrapped into a Flask server, allowing input to be received via a browser and appropriate results to be returned.

The screenshot displays a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'SKINCARE PRODUCT RECOMMENDATION' containing files like 'Beauty Match...', 'cosmetic_p.csv', and 'app.py'. The code editor shows the following Python code:

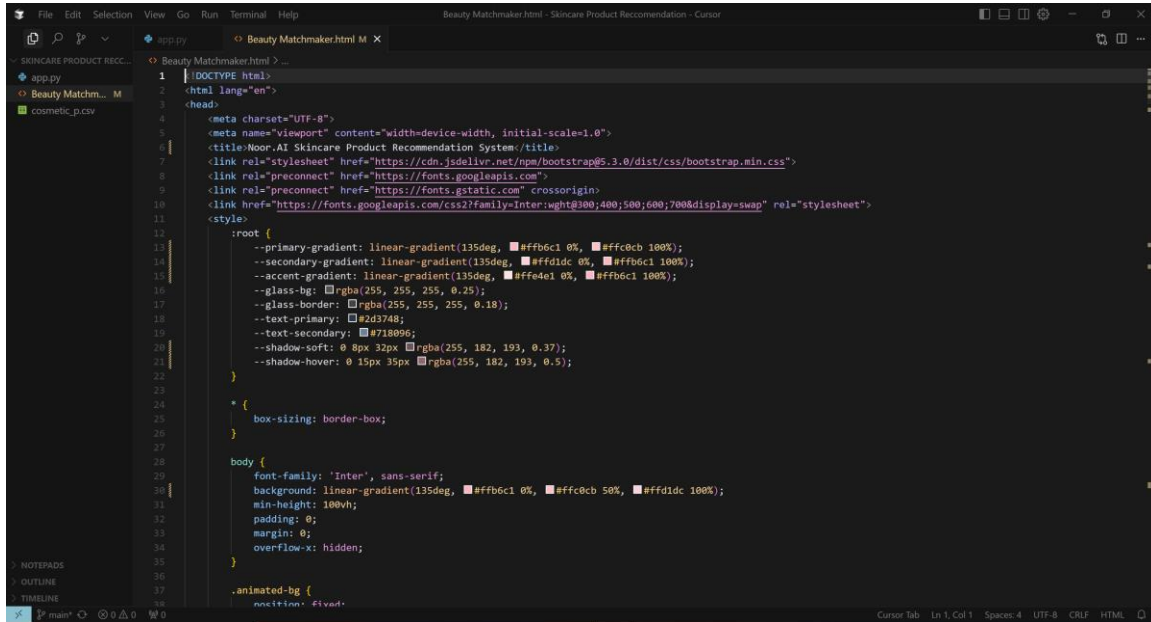
```
1 from flask import Flask, request, render_template_string
2 import pandas as pd
3
4 app = Flask(__name__)
5
6 # Load and clean the dataset once at startup
7 df = pd.read_csv("cosmetic_p.csv")
8
9 # Clean the data
10 df["Label"] = df["Label"].str.title().str.strip()
11 df = df.drop(columns=[col for col in ["Combination", "Sensitive"] if col in df.columns])
12 skin_cols = ["Dry", "Normal", "Oily"]
13 df["skin_type_coverage"] = df[skin_cols].sum(axis=1)
14 label_order = ["Moisturizer", "Cleanser", "Treatment", "Face Mask", "Eye Cream", "Sun Protect"]
15 df["Label"] = pd.Categorical(df["Label"], categories=label_order, ordered=True)
16
17 # Recommendation logic
18 def recommend_products(skin_type, budget=None, top_n_per_type=2):
19     filtered = df[(df[skin_type] == 1) & (df["skin_type_coverage"] >= 2)]
20     if budget:
21         filtered = filtered[filtered["price"] <= budget]
22     filtered_sorted = filtered.sort_values(by=["Label", "rank"], ascending=[True, False])
23     recommendations = filtered_sorted.groupby("Label").head(top_n_per_type)
24     return recommendations[["Label", "brand", "name", "price", "rank"]]
25
26 # Route
27 @app.route('/', methods=['GET', 'POST'])
28 def index():
29     recommendations = None
30     error = None
31
32     if request.method == "POST":
33         skin_type = request.form.get("skin_type", "").title()
34         budget_input = request.form.get("budget", "").strip()
35
36         if skin_type not in skin_cols:
37             error = "Invalid skin type selected."
38         else:
```

Noor.AI – Skincare Product Recommendation Feature Development Report

Created by Shafaq

Step 3: Frontend with HTML

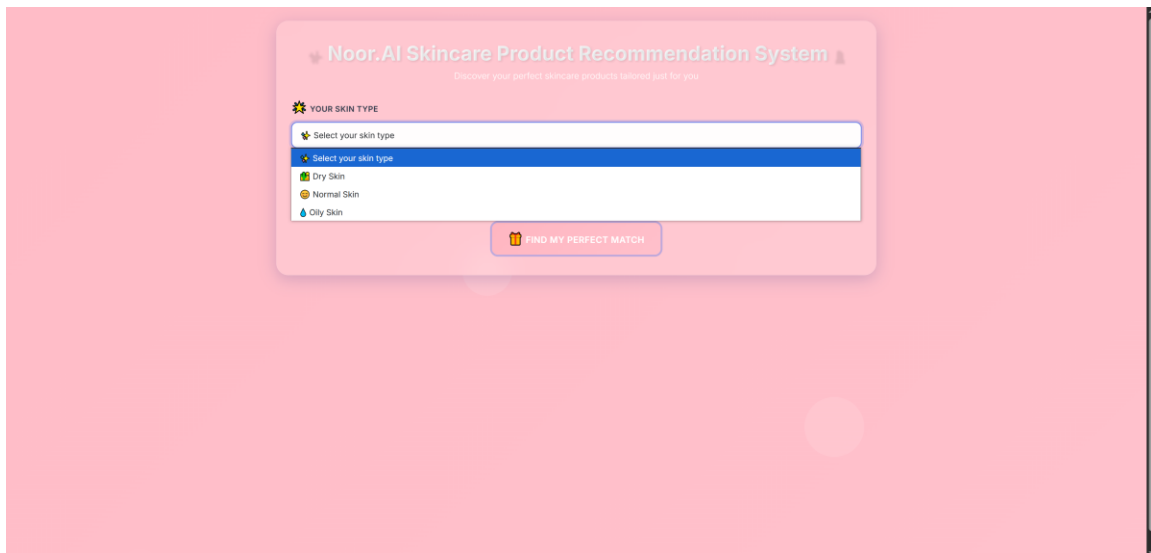
Claude.ai assisted in creating a basic HTML page featuring a dropdown for skin type selection. The HTML page connects to the Flask backend and displays recommended products.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Noor.AI Skincare Product Recommendation System</title>
7   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
8   <link rel="preconnect" href="https://fonts.googleapis.com">
9   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
10  <link href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700&display=swap" rel="stylesheet">
11 </head>
12 <body>
13   <!-- Primary Gradient -->
14   <!-- Secondary Gradient -->
15   <!-- Accent Gradient -->
16   <!-- Glass BG -->
17   <!-- Glass Border -->
18   <!-- Text Primary -->
19   <!-- Text Secondary -->
20   <!-- Shadow Soft -->
21   <!-- Shadow Hover -->
22 </body>
23 </html>
```

Step 4: End-to-End Testing

All components—selection input, backend logic, and frontend result display—were tested to ensure products for selected all different skin types were accurately recommended.



Noor.AI – Skincare Product Recommendation Feature Development Report

Created by Shafaq

Noor.AI Skincare Product Recommendation System
Discover your perfect skincare products tailored just for you

YOUR SKIN TYPE

Oily Skin

BUDGET (OPTIONAL)

Enter your budget (e.g., 50.00)

FIND MY PERFECT MATCH

Noor.AI Skincare Product Recommendation System
Discover your perfect skincare products tailored just for you

YOUR SKIN TYPE

Select your skin type

Dry Skin

Normal Skin

Oily Skin

FIND MY PERFECT MATCH

Your Personalized Recommendations

| LABEL | BRAND | NAME | PRICE | RANK |
|-------------|--------------------|--|-------|------|
| Moisturizer | CLINIQUE | Limited Edition Dramatically Different™ Moisturizing Gel | 39 | 5.0 |
| Moisturizer | LANEIGE | Water Bank Dual Layer Face Oil | 38 | 5.0 |
| Cleanser | KIEHL'S SINCE 1851 | Epidermal Re-Texturizing Micro-Dermabrasion | 41 | 5.0 |
| Cleanser | ERNO LASZLO | Pore Refining Detox Double Cleanse | 55 | 5.0 |
| Treatment | BIOEFFECT | EGF Serum | 160 | 5.0 |

9. Conclusion

This module was developed as a testable standalone feature, focusing on accurate skincare product recommendations based on user-selected skin types. Products tagged for all skin types are intelligently included to broaden recommendations. After successful development and validation, it is now ready for integration into the Noor.AI web application.