

# 1. Problem Formulation

## 1.1 Business Problem Definition

RecoMart is an e-commerce platform that aims to improve **customer engagement and conversion rate** by providing **personalized product recommendations** to its users.

Currently, users browse products without personalized guidance, which results in:

- Lower click-through rates
- Missed cross-selling opportunities
- Reduced average order value

### **Business Goal:**

Design a data-driven recommendation system that suggests relevant products to users based on their past behavior and product characteristics, thereby improving:

- Conversion rate
- User engagement
- Cross-selling effectiveness

The recommendation system must be continuously updated using fresh user interaction and transaction data.

---

## 1.2 Key Data Sources and Attributes

RecoMart collects data from multiple sources. The pipeline integrates the following key datasets:

### **1. User Interaction Data (Clickstream Logs)**

Captured from web and mobile platforms.

#### **Attributes:**

- `user_id` – Unique identifier for users
- `product_id` – Identifier of interacted product
- `event_type` – View, click, add-to-cart
- `timestamp` – Time of interaction
- `device` – Web or mobile

This data reflects **implicit user preferences**.

---

## 2. Transactional Purchase Data

Records confirmed purchases made by users.

**Attributes:**

- `transaction_id`
- `user_id`
- `product_id`
- `quantity`
- `price`
- `timestamp`

This data represents **explicit user intent and value-based interactions**.

---

## 3. Product Metadata (Catalog / API)

Fetched from internal or external product services.

**Attributes:**

- `product_id`
- `category`
- `brand`
- `price`
- `popularity_score`

This data supports **content-based recommendations** and feature enrichment.

---

## 1.3 Expected Outputs from the Pipeline

The end-to-end data pipeline is expected to generate the following outputs:

### 1. Cleaned and Validated Datasets

- Structured and validated datasets for exploratory data analysis (EDA)
- Removal of duplicates, handling missing values, and schema consistency

### 2. Engineered Feature Sets

Features suitable for recommendation algorithms, such as:

- User activity frequency
- Product popularity
- Average user-item interaction strength
- Aggregated behavioral statistics

These features support:

- Collaborative filtering

- Content-based recommendation models

### 3. Deployable Recommendation Model

- A trained recommendation model capable of generating personalized product suggestions
  - A simple inference interface to retrieve top-N product recommendations for a user
- 

## 1.4 Evaluation Metrics

The recommendation model will be evaluated using **ranking-based metrics**, which are standard for recommendation systems:

- **Precision@K**  
Measures the proportion of relevant items among the top-K recommended products.
- **Recall@K**  
Measures the proportion of relevant items successfully retrieved in the top-K recommendations.
- **Normalized Discounted Cumulative Gain (NDCG)**  
Evaluates ranking quality by assigning higher importance to correctly ranked items at higher positions.

These metrics align directly with the business objective of improving recommendation relevance and user engagement.

---

## 1.5 Expected Pipeline Outcomes

By implementing this pipeline, RecoMart will achieve:

- A scalable and automated data ingestion and processing system
- High-quality, versioned datasets for machine learning

- Reproducible feature generation for training and inference
- A recommendation model that learns continuously from fresh data

## 2. Data Collection and Ingestion

### 2.1 Data Sources Ingested

The pipeline ingests data from **multiple heterogeneous sources**, simulating a real-world e-commerce data ecosystem:

#### 1. User Interaction Data (Clickstream)

- **Source Type:** CSV files
- **Origin:** Web and mobile platforms
- **Ingestion Mode:** Batch (daily)

This data captures **implicit user behavior**, such as views, clicks, and add-to-cart events.

---

#### 2. Product Metadata

- **Source Type:** REST API

**API Endpoint:**

<https://fakestoreapi.com/products>

- 
- **Ingestion Mode:** Automated API-based ingestion

This API provides product catalog information including price, category, ratings, and descriptions, which supports **content-based feature generation**.

---

#### 2.2 Ingestion Design

Each ingestion script is designed following modern data engineering best practices.

## **Automated and Periodic Fetching**

- Ingestion scripts are designed to run periodically (e.g., daily)
  - Date-based folder partitioning enables incremental ingestion and replay
- 

## **Error Handling and Retry Mechanism**

- API ingestion includes retry logic to handle transient failures
  - HTTP errors and network issues are captured and retried up to a configurable limit
  - Failures do not corrupt previously ingested data
- 

## **Logging and Audit Trails**

- All ingestion activities are logged using a centralized logging mechanism
  - Logs capture:
    - Start and end of ingestion
    - Success or failure status
    - Retry attempts and error messages
- 

## **2.3 Ingestion Implementation**

### **Clickstream Data Ingestion**

- Reads interaction data from CSV files

Writes raw data into a timestamp-partitioned data lake structure:

```
data/raw/clickstream/YYYY/MM/DD/
```

- 
- 

## Product Metadata Ingestion

- Fetches product data from the Fake Store REST API
- Stores the **raw API response without transformation** to preserve source fidelity

Writes data to:

```
data/raw/products/YYYY/MM/DD/products.json
```

- 

Storing raw API responses ensures reproducibility and allows downstream transformations to be re-applied if needed.

---

## 2.4 Raw Data Storage Structure

```
data/raw/
└── clickstream/
    └── YYYY/MM/DD/clickstream.csv
└── products/
    └── YYYY/MM/DD/products.json
```

This structure mirrors cloud-based data lake designs and supports scalable downstream processing.

---

## 2.5 Logs Showing Ingestion Success and Failure

All ingestion runs generate logs stored at:

logs/ingestion.log

### **Successful Ingestion Example**

```
INFO - Starting product API ingestion
INFO - Product data ingested successfully from API at
data/raw/products/2025/01/01/products.json
```

### **Failure and Retry Example**

```
ERROR - Attempt 1 failed: Connection timeout
ERROR - Attempt 2 failed: Connection timeout
ERROR - Product ingestion failed after retries
```

---

## **2.6 Summary**

This ingestion layer:

- Integrates batch and API-based data sources
- Ensures fault tolerance through retries and logging
- Preserves raw data for lineage and reproducibility
- Provides a reliable foundation for downstream validation and feature engineering

# 3. Raw Data Storage

## 3.1 Storage Approach

The ingested data is stored in a **local filesystem-based data lake**, which simulates cloud object storage systems such as AWS S3.

Using the local filesystem allows the pipeline to be executed and validated without external dependencies while still following modern data lake design principles.

All storage paths are **centrally managed using configuration files**, ensuring that data locations are not hardcoded in the ingestion logic.

---

## 3.2 Data Lake Folder Structure

Raw data is organized using a structured, hierarchical folder layout based on:

- **Data source** (clickstream, products)
- **Ingestion date** (YYYY/MM/DD)

### Raw Data Layout

```
data/raw/
└── clickstream/
    ├── YYYY/
    │   └── MM/
    │       └── DD/
    │           └── clickstream.csv
    └── products/
        ├── YYYY/
        │   └── MM/
        │       └── DD/
        └── products.json
```

This layout mirrors industry-standard data lake structures used in large-scale data platforms.

---

### 3.3 Partitioning Strategy

A **date-based partitioning strategy** is used for raw data storage:

```
/<data_source>/<YYYY>/<MM>/<DD>/
```

#### Benefits:

- Enables incremental data processing
  - Supports historical reprocessing and backfills
  - Simplifies debugging and auditability
  - Preserves ingestion-time data lineage
- 

### 3.4 Storage Configuration

All storage locations are defined in a centralized configuration file:

 config/paths.yaml

```
raw: data/raw
validated: data/validated
processed: data/processed
features: data/features
logs: logs
source_files: data/source_files
```

Ingestion scripts dynamically resolve storage paths using this configuration, allowing the pipeline to remain flexible and environment-independent.

---

### 3.5 Data Upload Mechanism

During execution, ingestion scripts:

- Read source data from configured upstream locations
- Create date-partitioned directories automatically
- Write raw data files to the data lake
- Log storage paths and execution status

No manual upload steps are required, as data is written programmatically during ingestion.

---

## 3.6 Traceability and Reproducibility

- Raw data files are stored as **immutable artifacts**
- Each ingestion run creates a new, timestamp-partitioned directory
- Ingestion logs capture file paths and execution timestamps

This design ensures full traceability and allows downstream pipeline stages to be re-run reliably using historical raw data.

# Data Quality Report

This report summarizes data profiling and validation checks performed on raw datasets.

## Clickstream Data Validation Summary

Dataset	File	Rows	Missing Values	Duplicates	Invalid Events	Invalid Devices
clickstream	data\raw\clickstream\2026\01\10\clickstream.csv	61	0	0	-62	-62
clickstream	data\raw\clickstream\2026\01\20\clickstream.csv	61	0	0	-62	-62

## Product Data Validation Summary

Dataset	File	Rows	Missing Values	Duplicates	Invalid Price	Invalid Rating
Products	data\raw\products\2026\01\10\products.json	20	0	0	0	0
Products	data\raw\products\2026\01\20\products.json	20	0	0	0	0

# 5. Data Preparation and Exploratory Data Analysis

## 5.1 Preparation Approach

The data preparation stage transforms validated raw data into a **clean, structured, and machine-learning-ready dataset** suitable for feature engineering and recommendation modeling.

This stage focuses on:

- Cleaning and filtering interaction data
- Enriching user interactions with product metadata
- Encoding categorical attributes
- Normalizing numerical variables
- Generating reproducible exploratory analysis artifacts

All transformations are performed programmatically to ensure **consistency and reproducibility**.

---

## 5.2 Data Cleaning and Enrichment

The following cleaning steps are applied to the clickstream interaction data:

- Removal of records with missing `user_id` or `product_id`
- Filtering to retain valid interaction types (`view`, `click`, `add_to_cart`)
- Conversion of timestamps into standardized datetime format

User interaction data is then **enriched** by joining with product metadata using `product_id`, allowing interaction records to include product attributes such as category, price, and ratings.

---

## 5.3 Feature Encoding and Normalization

To prepare the dataset for downstream modeling, the following transformations are applied:

### Categorical Encoding

- Interaction types are mapped to numerical interaction strength values:
  - `view` = 1
  - `click` = 2
  - `add_to_cart` = 3
- Product categories are label-encoded into numerical identifiers

### Numerical Normalization

- Product prices are normalized using min–max scaling
- Temporal information is extracted from timestamps (hour of day) to capture time-based user behavior patterns

These steps ensure that all features are represented in a format suitable for machine learning algorithms.

---

## 5.4 Exploratory Data Analysis (EDA)

Exploratory analysis is conducted to understand the structure and characteristics of the prepared data. The following analyses are performed:

- **Interaction distribution** across event types

- **Item popularity analysis** based on user interactions
- **User-item sparsity analysis** to quantify the sparsity of the interaction matrix

All EDA visualizations are generated automatically and saved as reproducible artifacts.

---

## 5.5 Prepared Data and EDA Artifacts

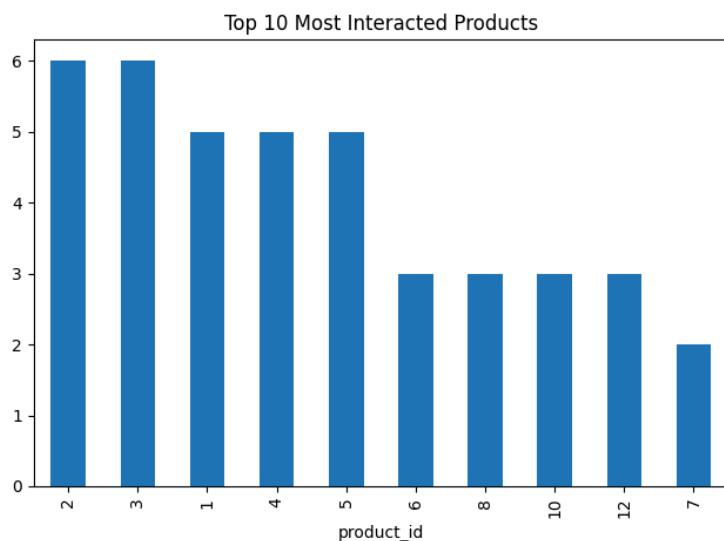
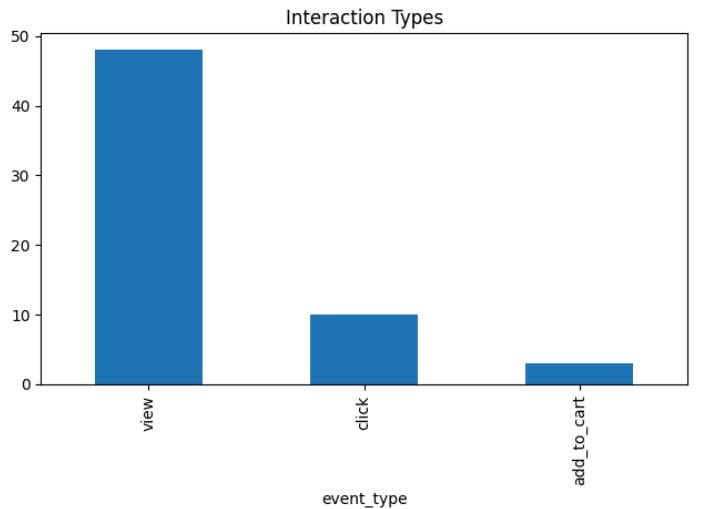
### Processed Data Output

`data/processed/prepared_interactions.csv`

This dataset contains cleaned and enriched interaction records and is ready for feature engineering and model training.

### EDA Artifacts

`data/processed/eda/`  
  └── `interaction_distribution.png`  
  └── `top_products.png`



These visualizations summarize key interaction patterns and are used for analysis and reporting.

---

## 5.6 Reproducibility and Pipeline Readiness

- All preparation steps are implemented as a standalone script
- Outputs are deterministic and reproducible
- Generated datasets and plots can be versioned and reused in downstream stages

This preparation layer provides a reliable foundation for **feature engineering and transformation** in the subsequent pipeline stage.

# 6. Feature Engineering and Transformation

## 6.1 SQL Schema

The following PostgreSQL schemas are used to store engineered features for the recommendation system. These tables act as a structured feature warehouse layer.

### User Features Table

```
CREATE TABLE user_features (
    user_id VARCHAR PRIMARY KEY,
    total_interactions INT,
    avg_interaction_score FLOAT
);
```

#### Description:

- Stores aggregated user-level behavioral features
  - Captures overall user activity and engagement strength
- 

### Item Features Table

```
CREATE TABLE item_features (
    product_id INT PRIMARY KEY,
    total_interactions INT,
    avg_interaction_score FLOAT,
    avg_rating FLOAT
);
```

#### Description:

- Stores item-level popularity and quality signals

- Supports popularity-based and content-aware recommendations
- 

## User–Item Interaction Features Table

```
CREATE TABLE user_item_features (
    user_id VARCHAR,
    product_id INT,
    interaction_count INT,
    total_interaction_score FLOAT,
    PRIMARY KEY (user_id, product_id)
);
```

### Description:

- Captures historical interaction strength between users and items
  - Forms the basis for collaborative filtering and matrix factorization models
- 

## 6.2 Transformation Scripts

Feature transformation is implemented using a standalone Python script:

### Script Location:

```
src/transformation/feature_engineering.py
```

### Responsibilities of the Script:

- Load prepared interaction data from  
`data/processed/prepared_interactions.csv`
- Aggregate user-level, item-level, and user–item features
- Connect to PostgreSQL using configuration-driven credentials

- Persist engineered features into relational tables

The script is designed to be **idempotent**, allowing safe re-execution during experimentation or pipeline reruns.

---

## 6.3 Summary of Feature Logic

### User-Level Features

- **Total Interactions:** Number of interactions performed by a user
- **Average Interaction Score:** Mean interaction strength across all user actions

Purpose:

- Represents user engagement intensity
  - Useful for user profiling and cold-start handling
- 

### Item-Level Features

- **Total Interactions:** Number of interactions received by an item
- **Average Interaction Score:** Mean engagement strength per item
- **Average Rating:** Mean product rating from metadata

Purpose:

- Captures item popularity and perceived quality
  - Supports popularity-based and hybrid recommendation models
- 

### User–Item Interaction Features

- **Interaction Count:** Number of times a user interacted with an item
- **Total Interaction Score:** Cumulative interaction strength

Purpose:

- Encodes historical preference strength
  - Directly usable in collaborative filtering algorithms
- 

## 6.4 Outcome

The engineered features are stored in a structured relational format and are ready for:

- Model training
- Inference-time feature retrieval
- Feature store abstraction in subsequent pipeline stages

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer    Dashboard X Properties X SQL X Statistics X Dependencies X Dependents X Processes X postgres/postgres@local\* X

Servers (1)    Databases (1)    Tables (1)

  local    postgres

  |--> Catalogs

  |--> Event Triggers

  |--> Extensions

  |--> Foreign Data Wrappers

  |--> Languages

  |--> Publications

  |--> Schemas (1)

    |--> public

    |--> Aggregates

    |--> Collations

    |--> Domains

    |--> FTS Configurations

    |--> FTS Dictionaries

    |--> FTS Functions

    |--> FTS Parsers

    |--> FTS Templates

    |--> Foreign Tables

    |--> Functions

    |--> Materialized Views

    |--> Procedures

    |--> Sequences

    |--> Tables

    |--> Trigger Functions

    |--> Types

    |--> Views

  |--> Subscriptions

  |--> Login/Group Roles

  |--> Tablespaces

Query    Query History

```

1 CREATE TABLE user_features (
2   user_id VARCHAR PRIMARY KEY,
3   total_interactions INT,
4   avg_interaction_score FLOAT
5 );
6
7
8 CREATE TABLE item_features (
9   product_id INT PRIMARY KEY,
10  total_interactions INT,
11  avg_interaction_score FLOAT,
12  avg_rating FLOAT
13 );
14
15
16 CREATE TABLE user_item_features (
17   user_id CHAR,
18   product_id INT,
19   interaction_count INT,
20   total_interaction_score FLOAT,
21   PRIMARY KEY (user_id, product_id)
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
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
308
309
309
310
311
312
313
313
314
315
316
316
317
318
319
319
320
321
322
323
323
324
325
326
326
327
328
329
329
330
331
332
332
333
334
335
335
336
337
338
338
339
339
340
341
341
342
343
343
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
14
```

public.user\_features/postgres/postgres@local

Query History

```
1 SELECT * FROM public.user_features
2
```

Data Output Messages Notifications

	user_id	total_interactions	avg_interaction_score
1	U1	5	1.6
2	U10	2	1
3	U11	3	1.3333333333333333
4	U12	2	1
5	U13	3	1.3333333333333333
6	U14	2	1
7	U15	3	1.6666666666666667
8	U16	2	1
9	U17	3	1.3333333333333333
10	U18	2	1
11	U19	2	1
12	U2	4	1.25
13	U20	4	1
14	U3	4	1.25
15	U4	4	1.25
16	U5	4	1.5
17	U6	3	1.3333333333333333
18	U7	3	1.3333333333333333
19	U8	3	1
20	U9	3	1.3333333333333333

Showing rows: 1 to 20 | Page No: 1 of 1 | << <> >> >>

✓ Successfully run. Total query runtime: 74 msec. 20 rows affected. ✘

✓ local/postgres - Database connected ✘ CRLF Ln 1, Col 1

Total rows: 20 Query complete 00:00:00.074

public.user\_item\_features/postgres/postgres@local

Query History

```
1 SELECT * FROM public.user_item_features
2
```

Data Output Messages Notifications

	user_id	product_id	interaction_count	total_interaction_score
1	U1	1	2	3
2	U1	2	2	4
3	U1	3	1	1
4	U10	5	1	1
5	U10	13	1	1
6	U11	6	2	3
7	U11	14	1	1
8	U12	7	1	1
9	U12	15	1	1
10	U13	8	2	3
11	U13	16	1	1
12	U14	9	1	1
13	U14	17	1	1
14	U15	10	2	4
15	U15	18	1	1
16	U16	11	1	1
17	U16	19	1	1
18	U17	12	2	3
19	U17	20	1	1
20	U18	13	1	1
21	U18	14	1	1
22	U19	15	1	1
23	U19	16	1	1
24	U2	3	2	3

Showing rows: 1 to 48 | Page No: 1 of 1 | << <> >> >>

CRLF Ln 1, Col 1

Total rows: 48 Query complete 00:00:00.104

The screenshot shows a PostgreSQL query editor interface. The query being run is:

```
1  SELECT * FROM public.item_features
```

The results are displayed in a table format:

	product_id	total_interactions	avg_interaction_score	avg_rating
1	1	5	1.4	3.9
2	2	6	1.5	4.1
3	3	6	1.333333333333333	4.7
4	4	5	1.6	2.1
5	5	5	1.2	4.6
6	6	3	1.333333333333333	3.9
7	7	2	1	3
8	8	3	1.333333333333333	1.8999999999999997
9	9	2	1	3.3
10	10	3	1.6666666666666667	2.9
11	11	2	1	4.8
12	12	3	1.333333333333333	4.8
13	13	2	1	2.9
14	14	2	1	2.2
15	15	2	1	2.6
16	16	2	1	2.9
17	17	2	1	3.8
18	18	2	1	4.7
19	19	2	1	4.5
20	20	2	1	3.6

Total rows: 20 Query complete 00:00:00.174 CRLF Ln 1, Col 1

This transformation layer bridges raw interaction data and recommendation models in a scalable and reproducible manner.

# 7. Feature Store

## 7.1 Feature Store Approach

A **custom feature store** is implemented to manage, document, and retrieve machine learning features in a consistent and versioned manner.

The feature store is designed to:

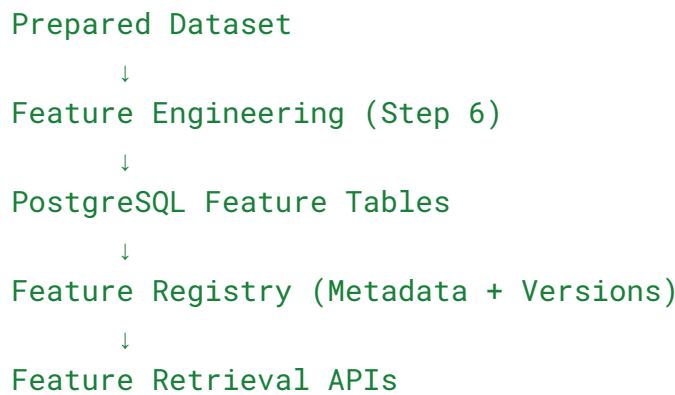
- Centralize access to engineered features
- Ensure consistency between training and inference
- Support feature versioning and metadata management

PostgreSQL is used as the **offline feature storage layer**, while a lightweight Python-based registry manages feature definitions and versions.

---

## 7.2 Feature Store Architecture

The feature store is built on top of the feature engineering pipeline and consists of the following components:



This architecture mirrors the design of production-grade feature stores while remaining simple and easy to maintain.

---

## 7.3 Feature Storage Layer

Engineered features are stored in PostgreSQL tables created during Step 6:

- `user_features`
- `item_features`
- `user_item_features`

---

These tables act as the **offline feature store**, enabling efficient querying and reuse across different stages of the ML pipeline.

---

## 7.4 Feature Metadata Registry

A centralized **feature registry** is used to document feature metadata and manage feature versions.

### Registry Location

`src/feature_store/registry.py`

The registry captures:

- Feature group (user, item, user-item)
- Feature names
- Source tables
- Feature descriptions
- Feature version identifiers

This ensures feature discoverability and traceability.

---

## 7.5 Feature Versioning Strategy

Feature versions are managed using explicit version identifiers (e.g., `v1`).

Each version:

- Represents a stable set of feature definitions
- Can be independently retrieved for training or inference
- Enables backward compatibility when features evolve

Versioning guarantees reproducibility of model training and evaluation.

---

## 7.6 Feature Retrieval Mechanism

Feature retrieval is implemented using reusable Python functions that query PostgreSQL based on the requested feature version.

Retrieval functions support:

- User-level feature lookup
- Item-level feature lookup
- User-item interaction feature lookup

The same retrieval logic is used during **model training and inference**, ensuring feature consistency across the ML lifecycle.

---

## 7.7 Sample Feature Retrieval Demonstration

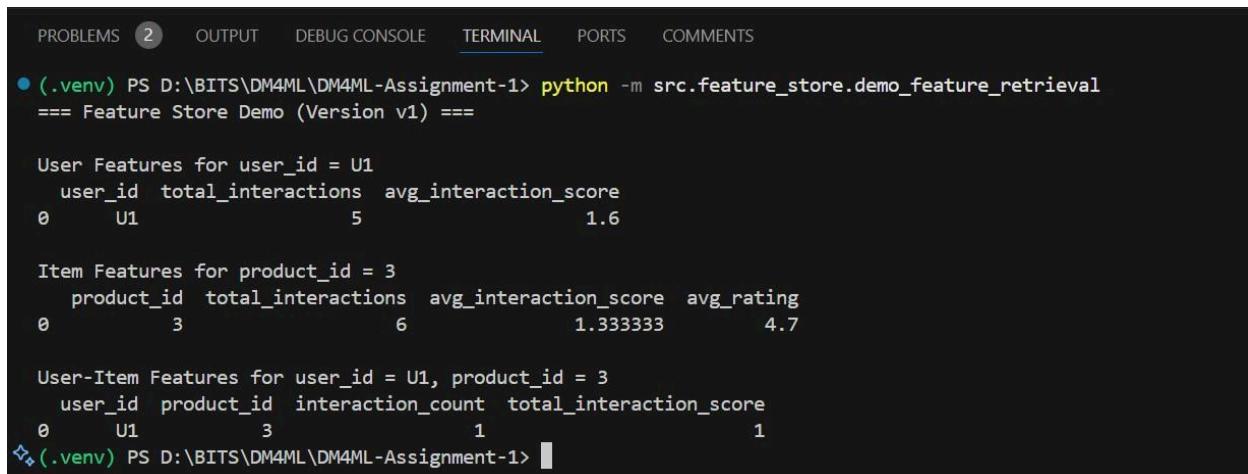
A standalone script demonstrates feature retrieval from the feature store.

### **Script Location**

`src/feature_store/demo_feature_retrieval.py`

The script:

- Retrieves user features for a given user ID
- Retrieves item features for a given product ID
- Retrieves user-item interaction features
- Uses versioned feature access (**v1**)



A screenshot of a terminal window titled "TERMINAL". The window shows the command "python -m src.feature\_store.demo\_feature\_retrieval" being run in a virtual environment named ".venv". The output displays three types of feature sets: User Features for user\_id = U1, Item Features for product\_id = 3, and User-Item Features for user\_id = U1, product\_id = 3. Each output is a table with columns like user\_id, total\_interactions, avg\_interaction\_score, etc.

```
● (.venv) PS D:\BITS\DM4ML\DM4ML-Assignment-1> python -m src.feature_store.demo_feature_retrieval
== Feature Store Demo (Version v1) ==

User Features for user_id = U1
  user_id  total_interactions  avg_interaction_score
  0        U1                  5                      1.6

Item Features for product_id = 3
  product_id  total_interactions  avg_interaction_score  avg_rating
  0            3                  6                  1.333333      4.7

User-Item Features for user_id = U1, product_id = 3
  user_id  product_id  interaction_count  total_interaction_score
  0        U1          3                  1                      1

◆ (.venv) PS D:\BITS\DM4ML\DM4ML-Assignment-1>
```

This demonstrates how downstream ML components can consume features reliably.

---

## 7.8 Training and Inference Consistency

- Both training and inference pipelines access features through the same APIs
- Feature definitions and transformations remain synchronized
- Versioned retrieval prevents feature drift between environments

This ensures reliable and repeatable model behavior.

---

## **7.9 Summary**

The feature store provides a structured, versioned, and documented mechanism for managing machine learning features. It bridges the gap between feature engineering and model consumption, enabling scalable and reproducible recommendation workflows.

# 8. Data Versioning and Lineage

## 8.1 Versioning Approach

Data versioning and lineage tracking are implemented using **DVC (Data Version Control)**. DVC enables version control of large datasets without storing the actual data files in the Git repository.

This approach ensures:

- Reproducibility of experiments
- Clear tracking of dataset evolution
- Separation of code and data versioning

Both raw and processed datasets are versioned to maintain end-to-end data lineage.

---

## 8.2 Datasets Under Version Control

The following data layers are tracked using DVC:

### Raw Data

`data/raw/`

This includes:

- Clickstream interaction data
- Product metadata ingested from the REST API

### Processed Data

`data/processed/`

This includes:

- Cleaned and prepared interaction datasets
- EDA artifacts such as plots and summaries

Tracking both layers ensures that transformations can always be traced back to the original data source.

---

## 8.3 DVC Initialization

DVC is initialized at the repository level using:

```
dvc init
```

This creates the required DVC configuration files:

```
.dvc/  
.dvcignore
```

These files are committed to Git to enable data versioning across the project.

---

## 8.4 Dataset Versioning Workflow

Raw and processed datasets are added to DVC tracking using:

```
dvc add data/raw  
dvc add data/processed
```

This generates lightweight metadata files:

```
data/raw.dvc  
data/processed.dvc
```

Only these `.dvc` files are committed to Git, while the actual data remains stored locally.

---

## 8.5 Repository Structure After Versioning

After enabling DVC, the repository structure includes:

```
DM4ML-Assignment-1/
├── data/
│   ├── raw/                      # Tracked by DVC
│   ├── processed/                # Tracked by DVC
│   ├── raw.dvc
│   └── processed.dvc
├── .dvc/
├── .dvcignore
└── src/
    ├── config/
    └── reports/
```

This structure clearly separates code, configuration, and versioned data artifacts.

---

## 8.6 Data Lineage Tracking

Data lineage is established through:

- Hierarchical data layers (raw → processed → features)
- Timestamp-based ingestion directories
- DVC metadata files that record dataset hashes and versions
- Git versioning of transformation scripts

This provides a clear mapping between:

data source → transformation logic → resulting datasets

---

## 8.7 Reproducibility Workflow

A specific version of the dataset can be restored using:

```
git checkout <commit-hash>
dvc checkout
```

This restores the exact versions of raw and processed datasets associated with that commit, ensuring reproducible data pipelines and experiments.

---

## 8.8 Summary

The use of DVC provides a reliable mechanism for data versioning and lineage tracking across the data management pipeline. It ensures that all datasets used for feature engineering and model training are traceable, reproducible, and aligned with their corresponding transformation logic.

# 9. Model Training and Evaluation

---

## 9.1 Training and Evaluation Scripts

Model training, evaluation, and inference are implemented as **separate, modular scripts** to reflect production-grade machine learning workflows.

### Training Script



**Location:**

`src/models/train_model.py`

#### Responsibilities:

- Reads user-item interaction features from the PostgreSQL feature store
  - Trains a collaborative filtering model using **Matrix Factorization (SVD)**
  - Stores the trained model as a serialized artifact
  - Logs training parameters and artifacts using MLflow
- 

### Evaluation Script



**Location:**

`src/models/evaluate_model.py`

#### Responsibilities:

- Loads the trained model artifact
- Evaluates the model using ranking-based metrics:

- Precision@K
  - Recall@K
  - Logs evaluation metrics to MLflow for experiment tracking
- 

## Inference Script

 **Location:**

`src/models/inference.py`

### Responsibilities:

- Loads the trained recommendation model
- Generates top-K product recommendations for a given user
- Demonstrates inference-time feature consumption

```
● (.venv) PS D:\BITS\DM4ML\DM4ML-Assignment-1> python -m src.models.train_model
2026/01/20 13:57:44 INFO mlflow.store.db.utils: Creating initial MLflow database tables...
2026/01/20 13:57:44 INFO mlflow.store.db.utils: Updating database tables
2026/01/20 13:57:44 INFO alembic.runtime.migration: Context impl SQLiteImpl.
2026/01/20 13:57:44 INFO alembic.runtime.migration: Will assume non-transactional DDL.
2026/01/20 13:57:44 INFO alembic.runtime.migration: Context impl SQLiteImpl.
2026/01/20 13:57:44 INFO alembic.runtime.migration: Will assume non-transactional DDL.
Model trained and saved successfully
● (.venv) PS D:\BITS\DM4ML\DM4ML-Assignment-1> python -m src.models.evaluate_model
2026/01/20 13:57:54 INFO mlflow.store.db.utils: Creating initial MLflow database tables...
2026/01/20 13:57:54 INFO mlflow.store.db.utils: Updating database tables
2026/01/20 13:57:54 INFO alembic.runtime.migration: Context impl SQLiteImpl.
2026/01/20 13:57:54 INFO alembic.runtime.migration: Will assume non-transactional DDL.
2026/01/20 13:57:54 INFO alembic.runtime.migration: Context impl SQLiteImpl.
2026/01/20 13:57:54 INFO alembic.runtime.migration: Will assume non-transactional DDL.
Precision@5: 0.1900
Recall@5: 0.9500
● (.venv) PS D:\BITS\DM4ML\DM4ML-Assignment-1> python -m src.models.inference
Recommendations for U1:
[ 2  1  3  7 11]
↖ (.venv) PS D:\BITS\DM4ML\DM4ML-Assignment-1>
```

---

## 9.2 Model Performance Report

Model performance is evaluated using **standard recommendation system metrics** that measure ranking quality.

### Evaluation Metrics

- **Precision@5:** Measures the relevance of the top-5 recommended items
- **Recall@5:** Measures the coverage of relevant items in the top-5 recommendations

### Performance Summary

A model performance report is generated based on the logged evaluation metrics and includes:

- Model type and configuration
- Training data source (PostgreSQL feature store)
- Precision@K and Recall@K scores
- Observations on model behavior and performance

These results are reproducible and traceable through MLflow experiment logs.

---

## 9.3 Tracked Model Metadata

All model-related metadata is tracked using **MLflow**, ensuring reproducibility and auditability.

### Tracked Information

- **Run IDs:** Unique identifiers for each experiment run
- **Parameters:** Model type, latent dimensions, data source
- **Metrics:** Precision@5, Recall@5

- **Artifacts:** Serialized trained model file

## Model Artifact

`models/svd_model.pkl`

## Experiment Tracking

MLflow provides a centralized UI to compare runs and inspect model performance:

`mlflow ui`

The screenshot displays two views of the MLflow UI.

**Top View (Home Page):**

- Header:** mlflow 3.8.1, GitHub, Docs
- Left Sidebar:** + New, Home, Experiments, Models, Prompts
- Information about UI telemetry:** MLflow collects usage data to improve the product. To confirm your preferences, please visit the settings page in the navigation sidebar. To learn more about what data is collected, please visit the documentation.
- Get started:**
  - Log traces: Trace LLM applications for debugging and monitoring.
  - Run evaluation: Iterate on quality with offline evaluations and comparisons.
  - Train models: Track experiments, parameters, and metrics throughout training.
  - Register prompts: Manage prompt updates and collaborate across teams.
- Experiments:**

Name	Time created	Last modified	Description	Tags
Default	01/20/2026, 01:57:11 PM	01/20/2026, 01:57:11 PM	-	
- Discover new features:**
  - MLflow MCP server: Connect your coding assistants and AI applications to MLflow and automatically analyze your experiments and traces.
  - Optimize prompts: Access the state-of-the-art prompt optimization algorithms such as MPROQ2, GPTA, through MLflow Prompt Registry.
  - Agents-as-a-judge: Leverage agents as a judge to perform deep trace analysis and improve your evaluation accuracy.
  - Dataset tracking: Track dataset lineage and versions and effectively drive the quality improvement loop.

**Bottom View (Machine Learning Run Details):**

- Header:** mlflow 3.8.1, GitHub, Docs
- Left Sidebar:** Default, Machine learning, Runs, Models, Traces
- Search Bar:** Search metric charts, metrics.mse < 1 and params.model = "tree"
- Run List:**
  - Run Name: capable-basis-319 (red dot)
  - Run Name: bony-worm-860 (orange dot)
  - Run Name: funny-pug-185 (pink dot)
- Metrics:**
  - Model metrics (2):**
    - precision\_at\_5: 0.19 (capable-basis-319)
    - recall\_at\_5: 0.95 (capable-basis-319)
  - System metrics (0):** No charts in this section. Click Add chart or drag and drop to add charts here.

## **9.4 Summary**

The model training and evaluation stage implements a complete and reproducible workflow for recommendation systems. Training, evaluation, and inference are clearly separated, performance metrics are computed using standard measures, and all model metadata is tracked using MLflow. This ensures transparency, repeatability, and alignment with modern MLOps best practices.

# 10. Pipeline Orchestration

## 10.1 Orchestration Approach

The end-to-end data and machine learning pipeline is orchestrated using **Prefect**, a Python-native workflow orchestration framework that runs seamlessly on Windows and Linux environments.

Prefect is used to:

- Automate pipeline execution
- Define task dependencies
- Provide task-level logging and retries
- Monitor pipeline execution status

This orchestration layer ensures reliable and repeatable execution of the complete recommendation system pipeline.

---

## 10.2 Pipeline Workflow

The pipeline is executed as a single flow with the following sequence of stages:

### Data Ingestion

- Data Validation
- Data Preparation and EDA
- Feature Engineering
- Feature Store Access
- Model Training
- Model Evaluation

Each stage corresponds to an independent task implemented in earlier pipeline steps.

---

## 10.3 Orchestration Implementation

The pipeline is implemented as a Prefect flow with multiple tasks, where each task invokes an existing processing script.

### Flow Definition

`src/orchestration/prefect_flow.py`

The flow coordinates execution of:

- Clickstream and product data ingestion
  - Data validation checks
  - Data preparation and exploratory analysis
  - Feature engineering and storage in PostgreSQL
  - Feature store access demonstration
  - Model training and evaluation
- 

## 10.4 Logging and Monitoring

Prefect provides built-in logging and monitoring capabilities:

- Each task logs execution start and completion status
- Failures are captured and retried automatically
- Downstream tasks are blocked on upstream task failures
- Execution status can be monitored via the Prefect UI or terminal logs

This enables effective observability and failure handling across the pipeline.

---

## 10.5 Execution and Verification

The pipeline is executed using:

```
python -m src.orchestration.prefect_flow
```

Successful execution is verified through:

- Console logs indicating completion of each task
- Prefect UI showing successful flow runs and task statuses

Screenshots or logs from successful executions are included as evidence of orchestration.

Flows / dm4ml\_recommendation\_pipeline

Flow Runs

2 total

Task Runs

8  
8 Completed 100%

Runs Deployments Details

2 Flow runs

dm4ml\_recommendation\_pipeline > aquamarine-albatross  
Completed 2026/01/20 02:25:11 PM 0 Parameters 13s 8 Task runs

dm4ml\_recommendation\_pipeline > judicious-seal  
Failed 2026/01/20 02:21:53 PM 0 Parameters 30s 1 Task run

Items per page 10

Ready to scale? Upgrade

Join the Community

Settings

This screenshot shows the 'Flows' section of the Prefect UI. It displays two flow runs: one completed run for the 'aquamarine-albatross' deployment and one failed run for the 'judicious-seal' deployment. The failed run includes detailed metrics like execution time and task counts. A sidebar on the left provides navigation links for various Prefect features.

Dashboard

Flow Runs

2 total

Task Runs

8  
8 Completed 100%

Active Work Pools

There are no active work pools to show. Any work pools you do have are paused.  
View all work pools

Ready to scale?

Webhooks, role and object-level security, and serverless push work pools on Prefect Cloud

Upgrade to Cloud

Ready to scale? Upgrade

Join the Community

Settings

This screenshot shows the main 'Dashboard' page. It highlights the 'Flow Runs' and 'Task Runs' sections, both of which show 8 items completed at 100%. Below these, there's a message about active work pools. A prominent 'Ready to scale?' banner at the bottom offers additional cloud features. The sidebar remains consistent with the previous screenshot, providing access to other Prefect tools.

```

prefect_flow.py 2, U pipeline.log M
logs > pipeline.log
1 2026-01-20 14:35:39,049 | INFO | prefect | Starting temporary server on http://127.0.0.1:8417
2 See https://docs.prefect.io/v3/concepts/server/how-to-guides for more information on running a dedicated Prefect server.
3 2026-01-20 14:35:39,049 | INFO | prefect | Starting temporary server on http://127.0.0.1:8417
4 See https://docs.prefect.io/v3/concepts/server/how-to-guides for more information on running a dedicated Prefect server.
5 2026-01-20 14:35:38,383 | INFO | httpx | HTTP Request: GET http://127.0.0.1:8417/api/admin/version "HTTP/1.1 200 OK"
6 2026-01-20 14:35:38,526 | INFO | httpx | HTTP Request: GET http://127.0.0.1:8417/api/carf-token?client=f2f25860-732c-40aa-b64e-946ef9e02373 "HTTP/1.1 422 Unprocessable Entity"
7 2026-01-20 14:35:38,536 | INFO | httpx | HTTP Request: GET http://127.0.0.1:8417/api/logs/ "HTTP/1.1 200 OK"
8 2026-01-20 14:35:38,546 | INFO | httpx | HTTP Request: POST http://127.0.0.1:8417/api/flows/ "HTTP/1.1 200 OK"
9 2026-01-20 14:35:38,566 | INFO | httpx | HTTP Request: POST http://127.0.0.1:8417/api/flow\_runs/ "HTTP/1.1 201 Created"
10 2026-01-20 14:35:38,599 | INFO | httpx | HTTP Request: POST http://127.0.0.1:8417/api/flow\_runs/38d5af0b-6646-4cf4-b61e-e2534f2e30c3/set\_state "HTTP/1.1 201 Create"
11 2026-01-20 14:35:38,604 | INFO | httpx | HTTP Request: GET http://127.0.0.1:8417/api/flow\_runs/38d5af0b-6646-4cf4-b61e-e2534f2e30c3 "HTTP/1.1 200 OK"
12 2026-01-20 14:35:38,618 | INFO | prefect:flow_runs | Beginning flow run 'esoteric-squirrel' for flow 'dm4ml_recommendation_pipeline'
13 2026-01-20 14:35:38,618 | INFO | prefect:flow_runs | Beginning flow run 'esoteric-squirrel' for flow 'dm4ml_recommendation_pipeline'
14 2026-01-20 14:35:38,831 | INFO | prefect:task_runs | Clickstream ingestion completed
15 2026-01-20 14:35:38,831 | INFO | prefect:task_runs | Clickstream ingestion completed
16 2026-01-20 14:35:38,831 | INFO | prefect:task_runs | Finished in state Completed()
17 2026-01-20 14:35:38,831 | INFO | prefect:task_runs | Finished in state Completed()
18 2026-01-20 14:35:38,901 | INFO | httpx | HTTP Request: GET http://127.0.0.1:8417/api/flows/ca1a7bbf-283f-4d2d-ac67-20f92b7bda4f "HTTP/1.1 200 OK"
19 2026-01-20 14:35:39,542 | INFO | prefect:task_runs | Product ingestion completed
20 2026-01-20 14:35:39,542 | INFO | prefect:task_runs | Product ingestion completed
21 2026-01-20 14:35:39,545 | INFO | prefect:task_runs | Finished in state Completed()
22 2026-01-20 14:35:39,545 | INFO | prefect:task_runs | Finished in state Completed()
23 2026-01-20 14:35:40,133 | INFO | prefect:task_runs | Data validation completed
24 2026-01-20 14:35:40,133 | INFO | prefect:task_runs | Data validation completed
25 2026-01-20 14:35:40,195 | INFO | prefect:task_runs | Finished in state Completed()
26 2026-01-20 14:35:40,195 | INFO | prefect:task_runs | Finished in state Completed()
27 2026-01-20 14:35:40,762 | INFO | httpx | HTTP Request: GET http://127.0.0.1:8417/api/csrf-token?client=2db7f49b-ed4a-4514-b044-0124a101e04b "HTTP/1.1 422 Unprocessable Entity"
28 2026-01-20 14:35:40,767 | INFO | httpx | HTTP Request: POST http://127.0.0.1:8417/api/logs/ "HTTP/1.1 201 Created"
29 2026-01-20 14:35:41,731 | INFO | prefect:task_runs | Data preparation completed
30 2026-01-20 14:35:41,731 | INFO | prefect:task_runs | Data preparation completed
31 2026-01-20 14:35:41,733 | INFO | prefect:task_runs | Finished in state Completed()
32 2026-01-20 14:35:41,733 | INFO | prefect:task_runs | Finished in state Completed()
33 2026-01-20 14:35:42,559 | INFO | prefect:task_runs | Feature engineering completed
34 2026-01-20 14:35:42,559 | INFO | prefect:task_runs | Feature engineering completed
35 2026-01-20 14:35:42,561 | INFO | prefect:task_runs | Finished in state Completed()
36 2026-01-20 14:35:42,561 | INFO | prefect:task_runs | Finished in state Completed()
37 2026-01-20 14:35:42,782 | INFO | httpx | HTTP Request: POST http://127.0.0.1:8417/api/logs/ "HTTP/1.1 201 Created"
38 2026-01-20 14:35:43,216 | INFO | prefect:task_runs | Feature store retrieval demo completed
39 2026-01-20 14:35:43,216 | INFO | prefect:task_runs | Feature store retrieval demo completed
40 2026-01-20 14:35:43,218 | INFO | prefect:task_runs | Finished in state Completed()
41 2026-01-20 14:35:43,218 | INFO | prefect:task_runs | Finished in state Completed()
42 2026-01-20 14:35:44,793 | INFO | httpx | HTTP Request: POST http://127.0.0.1:8417/api/logs/ "HTTP/1.1 201 Created"
43 2026-01-20 14:35:45,829 | INFO | prefect:task_runs | Model training completed
44 2026-01-20 14:35:45,829 | INFO | prefect:task_runs | Model training completed
45 2026-01-20 14:35:45,830 | INFO | prefect:task_runs | Finished in state Completed()
46 2026-01-20 14:35:45,830 | INFO | prefect:task_runs | Finished in state Completed()
47 2026-01-20 14:35:46,818 | INFO | httpx | HTTP Request: POST http://127.0.0.1:8417/api/logs/ "HTTP/1.1 201 Created"
48 2026-01-20 14:35:46,818 | INFO | prefect:task_runs | Model evaluation completed

```

## 10.6 Summary

The Prefect-based orchestration layer automates the entire data and machine learning workflow, ensuring structured execution, observability, and fault tolerance. This completes the end-to-end pipeline implementation for the recommendation system.