

Major Northeast Cities' Airbnb Listings Price Prediction

Yash Gudimalla, Pranavnath Bathula, Anish Jha



The Problem

The Issue:

- Airbnb pricing is highly subjective - hosts guess prices, and guests struggle to assess value
- Two Airbnbs can look identical in photos but have vastly different price tags with no apparent reason

The Goal:

- Build a data-driven system to predict the "fair" nightly price of listings in Major Northeast Cities
- Move from "gut feeling" to "quantitative assessment"

Pipeline (1/4) (Raw CSVs)

Raw Washington D.C. csv file

Raw NYC csv file

data = pd.read_csv('listings.csv')
1 id,listing_id,url,scrape_id,last_scraped,source_name,description,neighborhood_overview,picture_url,host_id,host_url,host_name,host_since,host_location,host_about,host_response
2 48824219,https://www.airbnb.com/rooms/48824219,20251008117547,2025-10-02,city_scrapes,Soho LES East Village private room downtown,,https://ao.muscache.com/pictures/14883316.htm
3 https://www.airbnb.com/rooms/4883316,20251008117547,2025-10-02,previous_scrape,Soho LES East Village private room downtown,,https://ao.muscache.com/pictures/14883316.htm
4 48837137,https://www.airbnb.com/rooms/48837137,20251008117547,2025-10-02,previous_scrape,Sunset Park - Quiet and close to subway!,Cozy, lovely bedroom with a comfortable
5 queen size bed, located in a quiet neighborhood, great location for exploring Brooklyn.,https://ao.muscache.com/pictures/148837137.htm
6 https://www.airbnb.com/rooms/48837137,20251008117547,2025-10-02,previous_scrape,Sunset Park - Quiet and close to subway!,Cozy, lovely bedroom with a comfortable
7 queen size bed, located in a quiet neighborhood, great location for exploring Brooklyn.,https://ao.muscache.com/pictures/148837137.htm
8 48839416,https://www.airbnb.com/rooms/48839416,20251008117547,2025-10-02,city_scrapes,Spacious 2 Bedrm Spacious Apartment near Manhattan.,This bed, furnished apt on the Bushwick,
9 Williamsburg, Brooklyn, NY 11205, United States.,https://ao.muscache.com/pictures/148839416.htm
10 https://www.airbnb.com/rooms/48844212,20251008117547,2025-10-02,previous_scrape,Beautiful apartment on quiet side of Bushwick.,Modern and cozy apartment in the Bushwick,
11 Williamsburg, Brooklyn, NY 11205, United States.,https://ao.muscache.com/pictures/148844212.htm
12 48842301,https://www.airbnb.com/rooms/48842301,20251008117547,2025-10-02,city_scrapes,Cozy room in Williamsburg.,This place is located in Williamsburg, close to popular
13 bars and restaurants.,https://ao.muscache.com/pictures/148842301.htm
14 Also a host & love traveling.,NA,NA,0%,https://ao.muscache.com/pictures/user/ed5f6198-bce2-47e7-8e77-d284a54e27.jpg?ak1l_policy=profile_small,https://ao.muscache.com/
15 https://www.airbnb.com/rooms/2595,20251008117547,2025-10-02,city_scrapes,Skylit Studio Oasis | Midtown Manhattan Sanctuary.,Prime Midtown | Spacious 580 Sq Ft | Pyramid
16 https://www.airbnb.com/rooms/6848,20251008117547,2025-10-02,city_scrapes,House of Oyo - A Historic Brownstone Mansion.,Listed on the prestigious St. Marks
17 https://www.airbnb.com/rooms/6848,20251008117547,2025-10-02,city_scrapes,House of Oyo - A Historic Brownstone Mansion.,Listed on the prestigious St. Marks
18 https://www.airbnb.com/rooms/6848,20251008117547,2025-10-02,city_scrapes,House of Oyo - A Historic Brownstone Mansion.,Listed on the prestigious St. Marks
19 I am a former Wall St professional turned Entrepreneur. I have a wonderful Soul and have been in 65 countries. Hoping to reach 100 by 75.
20
21 My motto: GoSeeDo!Share!
22
23 HouseOfOyo,a few days or more,%0%,f,https://ao.muscache.com/in/pictures/user/ed5f6198-bce2-47e7-8e77-d284a54e27.jpg?ak1l_policy=profile_small,https://ao.muscache.com/
24 2595,https://www.airbnb.com/rooms/2595,20251008117547,2025-10-02,city_scrapes,Skylit Studio Oasis | Midtown Manhattan Sanctuary.,Prime Midtown | Spacious 580 Sq Ft | Pyramid
25
26 I am a Sound Therapy Practitioner and Kundalini Yoga & Meditation teacher. I work with energy and sound for relaxation and healing, using Symphonic gong, singing bowls,
27 crystal bowls, Tibetan singing bowls, and various instruments.
28 Any questions, please text or call Jennifer at 646.498.8718. If there's a few days or more, 45%,24%,https://ao.muscache.com/in/pictures/user/ed5f6198-bce2-47e7-8e77-d284a54e27.jpg?ak1l_policy=profile_small,https://ao.muscache.com/
29 6848,https://www.airbnb.com/rooms/6848,20251008117547,2025-10-02,city_scrapes,city_scrapes,Only 2 steps to Manhattan studio.,Comfortable studio apartment with super comfortable king
30 Want to take a break from the city we'll tell you about wilderness canoeing on the Delaware river – just an hour and a half's drive. We can tell you where to find the gr
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
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
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
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
8010
8011
8012
8013
8014
8015
8016
8017
8018
8019
8020
8021
8022
8023
8024
8025
8026
8027
8028
8029
8030
8031
8032
8033
8034
8035
8036
8037
8038
8039
8040
8041
8042
8043
8044
8045
8046
8047
8048
8049
8050
8051
8052
8053
8054
8055
8056
8057
8058
8059
8060
8061
8062
8063
8064
8065
8066
8067
8068
8069
8070
8071
8072
8073
8074
8075
8076
8077
8078
8079
8080
8081
8082
8083
8084
8085
8086
8087
8088
8089
8090
8091
8092
8093
8094
8095
8096
8097
8098
8099
80100
80101
80102
80103
80104
80105
80106
80107
80108
80109
80110
80111
80112
80113
80114
80115
80116
80117
80118
80119
80120
80121
80122
80123
80124
80125
80126
80127
80128
80129
80130
80131
80132
80133
80134
80135
80136
80137
80138
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80199
80200
80201
80202
80203
80204
80205
80206
80207
80208
80209
80210
80211
80212
80213
80214
80215
80216
80217
80218
80219
80220
80221
80222
80223
80224
80225
80226
80227
80228
80229
80230
80231
80232
80233
80234
80235
80236
80237
80238
80239
80240
80241
80242
80243
80244
80245
80246
80247
80248
80249
80250
80251
80252
80253
80254
80255
80256
80257
80258
80259
80260
80261
80262
80263
80264
80265
80266
80267
80268
80269
80270
80271
80272
80273
80274
80275
80276
80277
80278
80279
80280
80281
80282
80283
80284
80285
80286
80287
80288
80289
80290
80291
80292
80293
80294
80295
80296
80297
80298
80299
80300
80301
80302
80303
80304
80305
80306
80307
80308
80309
80310
80311
80312
80313
80314
80315
80316
80317
80318
80319
80320
80321
80322
80323
80324
80325
80326
80327
80328
80329
80330
80331
80332
80333
80334
80335
80336
80337
80338
80339
80340
80341
80342
80343
80344
80345
80346
80347
80348
80349
80350
80351
80352
80353
80354
80355
80356
80357
80358
80359
80360
80361
80362
80363
80364
80365
80366
80367
80368
80369
80370
80371
80372
80373
80374
80375
80376
80377
80378
80379
80380
80381
80382
80383
80384
80385
80386
80387
80388
80389
80390
80391
80392
80393
80394
80395
80396
80397
80398
80399
80400
80401
80402
80403
80404
80405
80406
80407
80408
80409
80410
80411
80412
80413
80414
80415
80416
80417
80418
80419
80420
80421
80422
80423
80424
80425
80426
80427
80428
80429
80430
80431
80432
80433
80434
80435
80436
80437
80438
80439
80440
80441
80442
80443
80444
80445
80446
80447
80448
80449
80450
80451
80452
80453
80454
80455
80456
80457
80458
80459
80460
80461
80462
80463
80464
80465
80466
80467
80468
80469
80470
80471
80472
80473
80474
80475
80476
80477
80478
80479
80480
80481
80482
80483
80484
80485
80486
80487
80488
80489
80490
80491
80492
80493
80494
80495
80496
80497
80498
80499
80500
80501
80502
80503
80504
80505
80506
80507
80508
80509
80510
80511
80512
80513
80514
80515
80516
80517
80518
80519
80520
80521
80522
80523
80524
80525
80526
80527
80528
80529
80530
80531
80532
80533
80534
80535
80536
80537
80538
80539
80540
80541
80542
80543
80544
80545
80546
80547
80548
80549
80550
80551
80552
80553
80554
80555
80556
80557
80558
80559
80560
80561
80562
80563
80564
80565
80566
80567
80568
80569
80570
80571
80572
80573
80574
80575
80576
80577
80578
80579
80580
80581
80582
80583
80584
80585
80586
80587
80588
80589
80590
80591
80592
80593
80594
80595
80596
80597
80598
80599
80600
80601
80602
80603
80604
80605
80606
80607
80608
80609
80610
80611
80612
80613
80614
80615
80616
80617
80618
80619
80620
80621
80622
80623
80624
80625
80626
80627
80628
80629
80630
80631
80632
80633
80634
80635
80636
80637
80638
80639
80640
80641
80642
80643
80644
80645
80646
80647
80648
80649
80650
80651
80652
80653
80654
80655
80656
80657
80658
80659
80660
80661
80662
80663
80664
80665
80666
80667
80668
80669
80670
80671
80672
80673
80674
80675
80676
80677
80678
80679
80680
80681
80682
80683
80684
80685
80686
80687
80688
80689
80690
80691
80692
80693
80694
80695
80696
80697
80698
80699
80700
80701
80702
80703
80704
80705
80706
80707
80708
80709
80710
80711
80712
80713
80714
80715
80716
80717
80718
80719
80720
80721
80722
80723
80724
80725
80726
80727
80728
80729
80730
80731
80732
80733
80734
80735
80736
80737
80738
80739
80740
80741
80742
80743
80744
80745
80746
80747
80748
80749
80750
80751
80752
80753
80754
80755
80756
80757
80758
80759
80760
80761
80762
80763
80764
80765
80766
80767
80768
80769
80770
80771
80772
80773
80774
80775
80776
80777
80778
80779
80780
80781
80782
80783
80784
80785
80786
80787
80788
80789
80790
80791
80792
80793
80794
80795
80796
80797
80798
80799
80800
80801
80802
80803
80804
80805
80806
80807
80808
80809
80810
80811
80812
80813
80814
80815
80816
80817
80818
80819
80820
80821
80822
80823
80824
80825
80826
80827
80828
80829
80830
80831
80832
80833
80834
80835
80836
80837
80838
80839
80840
80841
80842
80843
80844
80845
80846
80847
80848
80849
80850
80851
80852
80853
80854
80855
80856
80857
80858
80859
80860
80861
80862
80863
80864
80865
80866
80867
80868
80869
80870
80871
80872
80873
80874
80875
80876
80877
80878
80879
80880
80881
80882
80883
80884
80885
80886
80887
80888
80889
80890
80891
80892
80893
80894
80895
80896
80897
80898
80899
80900
80901
80902
80903
80904
80905
80906
80907
80908
80909
80910
80911
80912
80913
80914
80915
80916
80917
80918
80919
80920
80921
80922
80923
80924
80925
80926
80927
80928
80929
80930
80931
80932
80933
80934
80935
80936
80937
80938
80939
80940
80941
80942
80943
80944
80945
80946
80947
80948
80949
80950
80951
80952
80953
80954
80955
80956
80957
80958
80959
80960
80961
80962
80963
80964
80965
80966
80967
80968
80969
80970
80971
80972
80973
80974
80975
80976
80977
80978
80979
80980
80981
80982
809

Raw Boston csv file

Pipeline (2/4) (Cleaning CSVs to SQL DB)

```
# Process each city's listings
all_cleaned_data = []

for city in cities:
    city_dir = data_dir / city
    listings_path = city_dir / "listings.csv"

    if not listings_path.exists():
        print(f"\nSkipping {city}: {listings_path} not found")
        continue

    print(f"\nProcessing {city}...")
    print(f"\nOriginal shape: {df.shape}")

    # Read the listings CSV
    df = pd.read_csv(listings_path)
    print(f"Original shape: {df.shape}")

    # Basic price cleaning
    df_clean = df.copy()

    # Remove $ and commas from price, convert to numeric
    price_str = (
        df_clean["price"]
        .astype(str)
        .str.replace("$", "", regex=False)
        .str.replace(",", "", regex=False)
        .str.strip()
    )

    df_clean["price"] = pd.to_numeric(price_str, errors="coerce")

    # Remove rows with missing or invalid prices
    original_count = len(df_clean)
    df_clean = df_clean[df_clean["price"].notna()]
    df_clean = df_clean[df_clean["price"] > 0]

    all_cleaned_data.append(df_clean)
```

notebooks/clean_csvs.ipynb



```
-- SQLite Schema for Airbnb Price Predictor
-- Enable foreign key constraints
PRAGMA foreign_keys = ON;

CREATE TABLE neighbourhood (
    neighbourhood_id INTEGER PRIMARY KEY AUTOINCREMENT,
    borough      TEXT,
    neighbourhood_name TEXT NOT NULL
);

CREATE TABLE listing (
    listing_id   INTEGER PRIMARY KEY,
    neighbourhood_id INTEGER REFERENCES neighbourhood(neighbourhood_id),
    city         TEXT, -- NYC, Boston, or Washington DC

    -- Host Information
    host_id      INTEGER,
    host_name    TEXT,
    host_since   DATE,
    host_is_superhost INTEGER, -- 0 = false, 1 = true

    -- Property Information
    room_type    TEXT,
    property_type TEXT,
    accommodates INTEGER,
    bedrooms     INTEGER,
    beds         INTEGER,
    bathrooms    REAL,
    bathrooms_text TEXT,

    latitude     REAL,
    longitude    REAL,

    -- Price and Availability
    price        REAL,
    number_of_reviews INTEGER,
    availability_365 INTEGER,
    estimated_revenue REAL,

    -- Review Information
    first_review DATE,
    last_review  DATE,
    review_scores_rating REAL,
    instant_bookable INTEGER, -- 0 = false, 1 = true
    calculated_host_listings_count INTEGER,
    reviews_per_month REAL
);
```

sql/schema/schema_sqlite.sql

This layer acts as the system's quality control. It changes raw, untyped CSVs into a structured relational database. By removing text artifacts, such as currency symbols, and applying strict SQL schema rules, we make sure that only clean, type-safe data goes to the modeling stage.

Pipeline (3/4) (Loading DB to Creating Features)

```
# Load all city data
all_city_data = {}

for city_config in cities_config:
    city_folder = city_config["folder"]
    city_name = city_config["name"]

    listings_file = processed_dir / f"{city_folder}.listings_cleaned.csv"
    neighbourhoods_file = data_dir / city_folder / "neighbourhoods.csv"

    if not listings_file.exists():
        print(f"\u25b6 Skipping {city_name}: {listings_file} not found")
        continue

    if not neighbourhoods_file.exists():
        print(f"\u25b6 Skipping {city_name}: {neighbourhoods_file} not found")
        continue

    print(f"\n{city_name}\n")
    print(f"\u25b6 Loading data for {city_name}...")
    print(f"\u25b6 {len(df_neighbourhoods)} neighbourhoods")

    # Load neighbourhoods
    print(f"\u25b6 Loading neighbourhoods from {neighbourhoods_file}...")
    df_neighbourhoods = pd.read_csv(neighbourhoods_file)
    print(f"\u25b6 Loaded {len(df_neighbourhoods)} neighbourhoods")

    # Load listings
    print(f"\u25b6 Loading listings from {listings_file}...")
    df_listings = pd.read_csv(listings_file)
    print(f"\u25b6 Loaded {len(df_listings)} listings")

    all_city_data[city_name] = {
        "neighbourhoods": df_neighbourhoods,
        "listings": df_listings,
        "folder": city_folder
    }
}
```

sql/etl/populate_database.ipynb



```
# 3.1 Target transform: log price (for later modeling)
df_features["log_price"] = np.log1p(df_features["price"])

# 3.2 Price per capacity features (ratios)
accom = df_features["accommodates"].replace(0, np.nan)
beds = df_features["beds"].replace(0, np.nan)
bedrooms = df_features["bedrooms"].replace(0, np.nan)

df_features["price_per_accommodate"] = df_features["price"] / accom
df_features["price_per_bed"] = df_features["price"] / beds
df_features["price_per_bedroom"] = df_features["price"] / bedrooms

# 3.3 Availability-based features
df_features["available_days_365"] = df_features["availability_365"]
df_features["availability_rate_365"] = df_features["available_days_365"] / 365.0
df_features["blocked_or_booked_days_365"] = 365 - df_features["available_days_365"]
df_features["blocked_or_booked_rate_365"] = (
    df_features["blocked_or_booked_days_365"] / 365.0
)

# 3.4 Review-based transforms
df_features["log_number_of_reviews"] = np.log1p(df_features["number_of_reviews"])
rpm = df_features["reviews_per_month"].clip(lower=0)
df_features["log_reviews_per_month"] = np.log1p(rpm)

# 3.5 Legacy features (for backward compatibility)
df_features["availability_ratio"] = df_features["availability_rate_365"]
df_features["is_high_rating"] = (df_features["review_scores_rating"] >= 4.8).astype(int)
df_features["is_active_host"] = (df_features["reviews_per_month"] > 0).astype(int)

df_features.head()
```

notebooks/features.ipynb

This step connects static storage with active modeling. After filling the centralized database with standardized records from multiple cities, we programmatically create valuable features, such as price-per-bedroom ratios and log-transformed metrics. These features reveal complex market patterns that the raw data alone cannot show.

Pipeline (4/4) (Using Features to Create Models and Predict)

```
num_features = numeric_features + binary_features
cat_features = categorical_features

numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scale", StandardScaler()),
])

categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore")),
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_features),
        ("cat", categorical_transformer, cat_features),
    ]
)

# Build the full pipeline: preprocessing + linear regression
linreg_model = Pipeline(steps=[
    ("preprocess", preprocess),
    ("regressor", LinearRegression())
])

# Fit the model on the training data
linreg_model.fit(X_train, y_train)

# Predict on the test data
y_pred_lr = linreg_model.predict(X_test)

# Evaluate metrics for linear regression
rmse_lr = sqrt(mean_squared_error(y_test, y_pred_lr))
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print("Global mean baseline RMSE: ", rmse_global)
print("Neighbourhood mean baseline RMSE: ", rmse_neigh)
print("Linear regression RMSE: ", rmse_lr)
print("Linear regression MAE: ", mae_lr)
print("Linear regression R^2: ", r2_lr)
```

Linear Regression model

```
# Random Forest Regressor (tree-based, non-linear model)
# Using the same preprocessing pipeline as linear regression
rf_model = Pipeline(steps=[
    ("preprocess", preprocess),
    ("regressor", RandomForestRegressor(
        n_estimators=200,
        max_depth=None,
        n_jobs=-1,
        random_state=42
    ))
]

# Fit on training data
rf_model.fit(X_train, y_train)

# Predict on test data
y_pred_rf = rf_model.predict(X_test)

# Evaluate metrics for Random Forest
rmse_rf = sqrt(mean_squared_error(y_test, y_pred_rf))
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest RMSE: ", rmse_rf)
print("Random Forest MAE: ", mae_rf)
print("Random Forest R^2: ", r2_rf)
```

Random Forest model

This final stage carries out the machine learning workflow by combining preprocessing, such as scaling and imputation, with modeling in one pipeline. We train both Linear Regression and Random Forest algorithms on our engineered features to create price predictions. We then compare their performance using metrics like RMSE and R².



Feature Engineering

- Started with the cleaned listing and neighbourhood tables. Constructed per-listing features (size, availability, reviews, host behaviour, log-transformed price and review counts) using a join.
- Developed neighbourhood and city-level context features (average ratings, number of listings, rates for superhosts, and entire homes) to constitute each listing's local market context.
- Bucketed continuous variables (capacity, host listings count, rating bands), encoded the room/property type and city as categorical variables; eliminated price-related leakage columns; and exported the feature engineered listings to an CSV file for use in developing models in the future.

```
city_env = (
    df_features
    .groupby("city")
    .agg(
        city_listing_count=("city", "size"),
        city_superhost_rate=("host_is_superhost", "mean"),
        city_avg_rating=("review_scores_rating", "mean"),
        city_avg_reviews_per_month=("reviews_per_month", "mean"),
    )
    .reset_index()
)

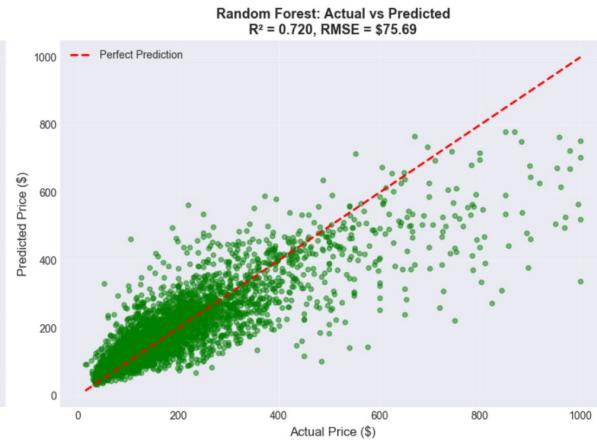
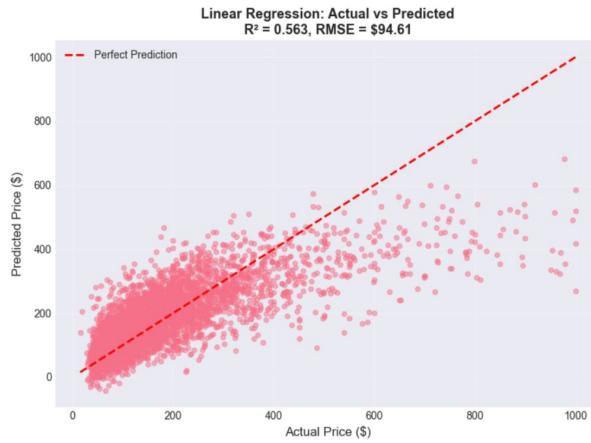
if "is_entire_home" in df_features.columns:
    entire_share = (
        df_features
        .groupby("city")["is_entire_home"]
        .mean()
        .rename("city_entire_home_share")
        .reset_index()
    )
    city_env = city_env.merge(entire_share, on="city", how="left")
else:
    entire_share = (
        df_features
        .assign(is_entire_home=(df_features["room_type"] == "Entire home/apt").astype(int))
        .groupby("city")["is_entire_home"]
        .mean()
        .rename("city_entire_home_share")
        .reset_index()
    )
    city_env = city_env.merge(entire_share, on="city", how="left")
```

Our Results & Performance

```
== Model comparison on test set ==
Global mean baseline RMSE: 143.09
Neighbourhood mean baseline RMSE: 124.57
Linear Regression RMSE: 94.61
Random Forest RMSE: 75.69

MAE scores:
Linear Regression MAE: 63.27
Random Forest MAE: 46.61

R^2 scores:
Linear Regression R^2: 0.563
Random Forest R^2: 0.720
```



Our analysis shows that the Random Forest Regressor is the best model. It achieved an R^2 of 0.720 and an RMSE of \$75.69, which is a 47% reduction in error compared to the neighborhood baseline. The scatter plots illustrate that the Random Forest predictions (green) cluster closely along the perfect prediction line. This demonstrates a much better ability to capture complex non-linear pricing factors than the scattered Linear Regression model.

Conclusion

This project shows the potential of combining data engineering, machine learning, and SQL to tackle important market challenges, such as evaluating fair property pricing in major Northeast cities: NYC, Boston, and Washington D.C. By carefully removing data leakage and focusing on valid features, our Random Forest model acts as a dependable support tool. It explains about 72% of pricing variance

(R^2 : 0.72) and achieves an RMSE of \$75.69, which is a 47% improvement over neighborhood baselines. In the end, this complete pipeline demonstrates how managing structured data, with over 20,000 listings in a normalized SQLite schema, and targeted algorithms can work together.

This effort creates repeatable solutions that directly enhance decision-making and transparency for both hosts and guests.