Streetwise Product Specification v2.0

Executive Summary

Streetwise is an AI-powered real estate valuation platform that provides prospective home buyers with standardized, data-driven analysis of New York Metro Area property listings. The platform combines machine learning, econometric modeling, and public data sources to generate a comprehensive 0-100 Streetwise Score representing a property's value-for-price relative to market conditions.

Core Value Proposition

- **For Buyers**: Democratizes access to professional-grade property analysis, closing the information asymmetry between buyers and sellers
- **Unique Approach**: Goes beyond simple comparables to quantify nuanced factors like renovation quality, layout efficiency, and hyper-local environmental conditions
- **Transparency**: Every score is explainable with clear attribution to specific factors and data sources

Product Architecture

1. Core Scoring Framework

1.1 Streetwise Score (0-100)

A composite score derived from five weighted categories, representing whether a listing offers good value relative to its asking price and inherent characteristics.

Score Interpretation Bands:

- **85-100**: Exceptional Value - Significantly underpriced relative to market
- **70-84**: Good Value - Fairly priced with positive attributes
- **50-69**: Average Value - Market-appropriate pricing
- **30-49**: Poor Value - Overpriced relative to comparables
- **0-29**: Significantly Overpriced - Major pricing concerns

1.2 Category Breakdown

Each category receives its own 0-100 score with detailed explainers:

Category	Weight	Description
Fair Value & Market Context	40%	Asking price vs. comp-adjusted expected price
Location & Neighborhood	20%	Transit access, schools, noise, amenities
Building & Amenities	15%	Building quality, amenities, services
Unit & Layout	20%	Renovation, features, layout efficiency
Bonuses/Penalties	±5-15%	Special conditions and deal-breakers

1.3 Confidence System

Every score includes a confidence metric (0-100%) based on:

- Data completeness (missing fields, quality of sources)
- Comparable properties count and similarity
- Model prediction intervals
- Data source reliability

Confidence-Based Display:

- **80-100% confidence**: "Streetwise Score" (standard display)
- **60-79% confidence**: "Preliminary Score" (with missing data indicators)
- **Below 60% confidence**: "Limited Analysis" (heavily caveated, optional display)

2. Data Pipeline & Ingestion

2.1 Data Sources Hierarchy

Primary Sources:

1. **StreetEasy Listing Data** (scraped via Firecrawl/Playwright)
    - Listing details, building information, photos, floorplans
    - Broker remarks and descriptions
    - Days on market, price history
2. **NYC Public Datasets:**
    - **DOF/ACRIS**: Closed sales transactions, deed records
    - **PLUTO**: Building metadata, lot characteristics
    - **DOE**: School zones, quality reports
    - **MTA GTFS**: Subway entrance locations
    - **311 Data**: Noise complaints (filtered for chronic issues)

- **OpenStreetMap**: POIs, street classifications

3. **Fallback: Manual Entry**
   - User-provided corrections and missing data
   - Validation against reasonable ranges

2.2 Ingestion Flow

```
User Input (URL/Address)
  ↓
Primary Scraping (Firecrawl)
  ↓
Success? → Parse to Schema
  ↓ No
Extension Fallback (if installed)
  ↓ Still incomplete
Manual Input Form (critical fields)
  ↓
Feature Engineering
  ↓
Scoring Pipeline
```

2.3 Data Schema

```json
```

```
{
  "listing": {
    "address": string,
    "unit": string,
    "price": number,
    "status": enum,
    "days_on_market": number,
    "price_history": array
  },
  "physical": {
    "bedrooms": number,
    "bathrooms": number,
    "square_feet": number | estimated,
    "floor": number,
    "total_floors": number,
    "outdoor_space": {
      "type": enum,
      "square_feet": number
    }
  },
  "financial": {
    "maintenance": number,
    "common_charges": number,
    "taxes": number,
    "assessment": number
  },
  "features": {
    "renovation_level": enum,
    "in_unit": array,
    "exposures": array,
    "views": array
  },
  "building": {
    "type": enum,
    "year_built": number,
    "units": number,
    "amenities": array
  },
  "location": {
    "coordinates": [lat, lng],
    "neighborhood": string,
    "school_zone": string,
    "subway_distance": number,
```

```
    "noise_score": number
  }
}
```

3. AI/ML Components

3.1 LLM-Powered Extraction

Purpose: Parse unstructured listing text and HTML into structured schema **Implementation**:

- GPT-4 class model with few-shot prompting
- Typed output validation against schema
- Confidence scoring for extracted fields

3.2 Renovation Level Classifier

Status: Implemented (96% accuracy) **Approach**: TF-IDF + Logistic Regression on text features **Classes**:

- Estate (needs full gut)
- Dated (functional but old)
- Recently Updated (refreshed <10 years)
- Gut High-End (premium renovation)
- New Development (never occupied)

3.3 Light Quality Classifier

Status: Roadmap (pending photo dataset) **Target Approach**: Multimodal (text + images) **Classes**: Poor, Average, Good, Exceptional **Current Performance**: 59% accuracy (text-only)

3.4 Layout Efficiency Analyzer

Status: Roadmap **Approach**:

- Floorplan detection from listing images
- OCR + vectorization to extract room polygons
- Calculate circulation vs. usable space ratios

4. Pricing Model & Comparables

4.1 Hedonic Pricing Model

Objective: Generate expected price using market comparables and property characteristics

Methodology:

```python
# Segment definition
segment = {
    'borough': categorical,
    'neighborhood_micro': categorical,
    'property_type': ['coop', 'condo', 'townhouse'],
    'bed_bucket': [0, 1, 2, 3, '4+'],
    'era': ['prewar', 'postwar', 'new_dev']
}

# Model specification
log(price) = β₀ + Σ(βᵢ × features) + segment_interactions + ε

# Features include all physical, location, and building characteristics
# Trained via Ridge/ElasticNet with cross-validation
```

Square Footage Policy:

- When square footage is missing (common in co-ops):
  1. Estimate using segment medians: `{borough × neighborhood × property_type × bed_bucket × era}`
  2. Maintain confidence score but flag as estimated
  3. Do NOT penalize the overall score
  4. Display transparency note in UI

4.2 Comparable Selection

Algorithm:

1. Find properties in same segment
2. Apply similarity scoring:
   - Same building line (highest weight)
   - Distance decay (exponential, τ = 0.4 miles)
   - Recency decay (linear, 18-month window)
3. Select top 10-20 comparables
4. Calculate adjustments for differences

5. Scoring Algorithms

5.1 Fair Value & Market Context (40%)

```python
def calculate_fair_value_score(listing, comps, market):
    expected_price = hedonic_model.predict(listing.features)
    price_gap = (expected_price - listing.price) / expected_price

    # Map to 0-100 scale with S-curve
    if price_gap > 0:  # Underpriced
        score = 50 + logistic_transform(price_gap, midpoint=0.08, slope=25)
    else:  # Overpriced
        score = 50 - logistic_transform(abs(price_gap), midpoint=0.08, slope=25)

    # Apply market context adjustments
    score *= market_adjustment_factor(listing.dom, listing.price_cuts)

    return clamp(score, 0, 100)
```

5.2 Location & Neighborhood (20%)

Sub-components:

Factor	Weight	Transform
Subway Distance	4%	Logistic (saturates at 15 min)
School Zone Rating	3%	Borough-relative percentile
Parks/Waterfront	3%	Logistic (saturates at 10 min)
Neighborhood Prestige	5%	StreetEasy taxonomy ranking
Noise Score	2%	Composite (see Noise Proxy section)
Block Quality	3%	Mid-block bonus, corner penalty

5.3 Noise Proxy Calculation

```python
```

```python
def calculate_noise_score(location):
    # Component weights
    traffic_weight = 0.40
    complaints_weight = 0.15
    nightlife_weight = 0.25
    emergency_weight = 0.20

    # Traffic proximity (to major roads)
    traffic_score = 100 * (1 - logistic(distance_to_primary, midpoint=40, k=0.1))

    # 311 complaints (chronic only, exclude construction)
    included_types = ['Noise - Street/Sidewalk', 'Noise - Commercial',
                      'Noise - Vehicle', 'Noise - Air Condition']
    complaints = filter_311(location, radius=100m, types=included_types)
    complaint_score = 100 * (1 - min(complaints/200, 1))

    # Nightlife density
    venues = count_pois(location, radius=90m, types=['bar', 'nightclub'])
    nightlife_score = 100 * (1 - sqrt(min(venues/10, 1)))

    # Emergency services
    emergency_dist = distance_to_nearest(['hospital', 'fire_station'])
    emergency_score = 100 * logistic(emergency_dist, midpoint=75, k=0.05)

    return weighted_average([traffic_score, complaint_score,
                            nightlife_score, emergency_score],
                           [traffic_weight, complaints_weight,
                            nightlife_weight, emergency_weight])
```

5.4 Building & Amenities (15%)

Scoring Matrix:

python

```
amenity_points = {
    'doorman': 2.0,
    'elevator': 2.5,
    'gym': 1.5,
    'pool': 1.0,
    'roof_deck': 1.5,
    'garage': 1.0,
    'storage': 0.5,
    'bike_room': 0.5,
    'laundry': 1.0,
    'live_in_super': 1.0
}

def calculate_building_score(building):
    base_score = 50

    # Amenity scoring with diminishing returns
    amenity_sum = sum([amenity_points.get(a, 0) for a in building.amenities])
    amenity_score = sqrt(min(amenity_sum / 10, 1)) * 30

    # Building quality adjustment
    if building.type == 'luxury':
        quality_bonus = 10
    elif building.type == 'walkup':
        quality_bonus = -5
    else:
        quality_bonus = 0

    # Maintenance fee assessment
    maint_percentile = calculate_percentile(building.maintenance_psf, segment)
    maint_score = (100 - maint_percentile) * 0.1

    return base_score + amenity_score + quality_bonus + maint_score
```

5.5 Unit & Layout (20%)

Components:

- Renovation Level: 7% (from classifier)
- In-Unit Features: 5% (W/D, dishwasher, central air, fireplace)
- Layout Efficiency: 3% (when floorplan available)
- Floor Height: 2% (percentile within building)

- Outdoor Space: 3% (with diminishing returns on size)

5.6 Bonuses & Penalties (±5-15%)

```python
adjustments = {
    # Penalties
    'ongoing_assessment': -5,
    'pending_litigation': -4,
    'estate_sale': -3,
    'flip_tax_high': -2,
    'no_pets': -2,
    'no_subletting': -3,

    # Bonuses
    'low_carrying_costs': +3,
    'tax_abatement': +2,
    'sponsor_sale': +1,
    'pets_allowed': +2,
    'private_terrace': +3,
    'river_views': +4
}
```

6. User Experience

6.1 Input Flow

Option A: URL Input (Primary)

```
1. User pastes StreetEasy URL
2. System attempts extraction via Firecrawl
3. If confidence < 70%, prompt for manual corrections
4. Generate and display score
```

Option B: Address Search

```
1. User enters address
2. Geocode and attempt to match listing
3. If no listing found, use public records only
4. Display limited analysis with manual input option
```

6.2 Results Display

Primary View:

- Overall Streetwise Score with confidence indicator
- Score interpretation (Exceptional/Good/Average/Poor Value)
- Price verdict with confidence level

Detailed Breakdown:

- Five category scores with meter visualizations
- Top positive and negative drivers
- Comparable properties map and table
- Market context charts (neighborhood trends, inventory)

Explainability Panel:

- Data sources for each factor
- Missing data indicators
- Calculation transparency
- "What-if" simulator for price changes

6.3 Manual Input Validation

```python
```

```
validation_rules = {
    'square_feet': {
        'min': 200,
        'max': 10000,
        'warning_if_outside': [
            segment_percentile(5),
            segment_percentile(95)
        ]
    },
    'price': {
        'min': 50000,
        'max': 50000000,
        'cross_check': 'price_per_sqft in [50, 5000]'
    },
    'maintenance': {
        'max': price * 0.05,  # Flag if >5% of price monthly
        'typical_range': segment_percentiles(10, 90)
    }
}
```

7. Technical Implementation

7.1 Technology Stack

Frontend:

- React + TypeScript
- Tailwind CSS
- Vite build system
- Mapbox/Leaflet for visualizations

Backend:

- Node.js edge functions (orchestration)
- Python FastAPI microservices (ML/pricing)
- PostgreSQL with PostGIS (spatial queries)
- Redis (caching layer)

Data Pipeline:

- Firecrawl/Playwright (scraping)

- Apache Airflow (ETL orchestration)
- dbt (data transformation)

ML Infrastructure:

- scikit-learn (baseline models)
- XGBoost (hedonic pricing)
- Weights & Biases (experiment tracking)

7.2 API Specification

```yaml
```

```yaml
endpoints:
  /api/v1/score:
    method: POST
    input:
      url: string | null
      address: string | null
      manual_overrides: object | null
    output:
      score: number
      confidence: number
      categories: object
      comparables: array
      explanations: object

  /api/v1/comparables:
    method: GET
    params:
      address: string
      radius: number
      limit: number
    output:
      properties: array
      statistics: object

  /api/v1/market:
    method: GET
    params:
      neighborhood: string
      timeframe: string
    output:
      median_prices: array
      inventory: array
      absorption_rate: number
```

8. Geographic Expansion Framework

8.1 Market Adaptation Requirements

Data Requirements by Market:

1. **Transaction Data**: MLS or public records access
2. **Transit Data**: GTFS feeds or equivalent

3. **School Data**: State education department APIs
4. **Neighborhood Definitions**: Local taxonomy mapping
5. **Market Characteristics**: Seasonality, typical DOM, price dynamics

8.2 Model Adaptation Process

```python
class MarketAdapter:
    def __init__(self, market_code):
        self.market = market_code
        self.load_market_config()

    def adapt_features(self):
        # Market-specific feature engineering
        if self.market == 'SF':
            self.features.add('earthquake_risk')
            self.features.add('tech_shuttle_access')
        elif self.market == 'MIA':
            self.features.add('flood_zone')
            self.features.add('hurricane_rating')

    def adapt_weights(self):
        # Market-specific category weights
        market_weights = {
            'NYC': {'location': 0.20, 'building': 0.15},
            'SF': {'location': 0.25, 'building': 0.10},
            'MIA': {'location': 0.15, 'building': 0.20}
        }
        return market_weights[self.market]
```

9. Personalization System

9.1 Dual Score Architecture

Standard Score: Market-objective valuation (unchangeable) **Fit Score**: Personalized based on user preferences

```python
```

```
def calculate_fit_score(listing, user_preferences):
    # Start with standard weights
    weights = copy.deepcopy(STANDARD_WEIGHTS)

    # Apply preference adjustments (capped at ±25% per factor)
    for pref, importance in user_preferences.items():
        affected_factors = PREFERENCE_MAPPING[pref]
        for factor in affected_factors:
            delta = importance * 0.25 / 3  # Scale 0-3 to weight delta
            weights[factor] *= (1 + delta)

    # Never adjust Fair Value weight
    weights['fair_value'] = STANDARD_WEIGHTS['fair_value']

    # Renormalize
    total = sum(weights.values())
    weights = {k: v/total for k, v in weights.items()}

    return calculate_score(listing, weights)
```

9.2 Preference Schema

```json
{
  "hard_requirements": {
    "pets_allowed": boolean,
    "minimum_beds": number,
    "maximum_price": number,
    "elevator": boolean
  },
  "soft_preferences": {
    "gym_importance": 0-3,
    "outdoor_space": 0-3,
    "quiet_street": 0-3,
    "school_quality": 0-3,
    "prewar_charm": 0-3
  }
}
```

10. Quality Assurance & Monitoring

10.1 Model Performance Metrics

Backtesting Framework:

```python
def backtest_model(time_window):
    # For each closed sale in window
    for sale in closed_sales:
        # Calculate score using only pre-sale data
        predicted_score = calculate_score(sale, date=sale.list_date)

        # Compare to actual outcome
        actual_discount = (sale.list_price - sale.sale_price) / sale.list_price

        # Track calibration
        calibration_metrics.add(predicted_score, actual_discount)

    return {
        'correlation': pearson_r(predicted_scores, actual_discounts),
        'mae': mean_absolute_error(predicted_prices, sale_prices),
        'calibration_plot': calibration_metrics.plot()
    }
```

Monitoring Dashboard:

- Score distribution by segment
- Confidence levels over time
- Data completeness rates
- Scraping success rates
- Model drift indicators

10.2 A/B Testing Framework

```python
```

```python
class ABTest:
    def __init__(self, feature_name):
        self.feature = feature_name
        self.control_model = current_model
        self.treatment_model = proposed_model

    def assign_user(self, user_id):
        # Deterministic assignment based on hash
        return 'treatment' if hash(user_id) % 2 == 0 else 'control'

    def track_metrics(self):
        return {
            'engagement': clicks_per_user,
            'accuracy': backtest_performance,
            'user_satisfaction': feedback_scores
        }
```

11. Roadmap & Future Features

Phase 1 (MVP - Current)

- ✅ Core scoring algorithm
- ✅ StreetEasy scraping
- ✅ Basic comparables
- ✅ Renovation classifier
- ✅ Web interface

Phase 2 (Q2 2025)

- [ ] Chrome extension
- [ ] User accounts & saved searches
- [ ] Personalized Fit Scores
- [ ] Enhanced comparables with photos
- [ ] What-if analysis tools

Phase 3 (Q3 2025)

- [ ] Light quality classifier (with photo dataset)
- [ ] Layout efficiency analyzer
- [ ] Empirically-derived weights (HBW)
- [ ] Advanced market predictions

- [ ] API for partners

Phase 4 (Q4 2025)

- [ ] Geographic expansion (SF, Boston, DC)
- [ ] Zillow/Redfin support
- [ ] Mobile applications
- [ ] Predictive sale price modeling
- [ ] Investment analysis tools

12. Compliance & Legal

12.1 Data Usage Guidelines

- Respect robots.txt and terms of service
- Implement rate limiting and backoff strategies
- Cache aggressively to minimize requests
- Provide user-agent identification

12.2 Disclaimers

- Not a professional appraisal
- For informational purposes only
- Users should verify all information independently
- Past performance doesn't guarantee future results

12.3 Privacy Policy Requirements

- No PII storage without consent
- Encrypted data transmission
- GDPR/CCPA compliance framework
- Data retention policies

13. Appendices

Appendix A: Feature Dictionary

Fair Value & Market Context Features (40% total weight)

Feature	Description	Weight	Transform
asking_vs_expected_price	Comp-adjusted price differential	15%	S-curve normalization
market_drift	YoY neighborhood median trends	5%	Linear scaling
price_per_sqft	Relative to neighborhood baseline	5%	Percentile ranking
days_on_market	Normalized vs neighborhood average	5%	Exponential decay
price_cut_history	Number and magnitude of cuts	3%	Cumulative penalty
seasonal_effects	Listing season adjustment	3%	Seasonal multiplier
absorption_rate	Local supply-demand balance	2%	Linear scaling
original_ask_discount	Current vs original list	2%	Linear scaling

Location & Neighborhood Features (20% total weight)

Feature	Description	Weight	Transform
neighborhood_prestige	StreetEasy neighborhood tier	5%	Categorical ranking
subway_distance	Walking minutes to nearest	4%	Logistic (saturates 15 min)
park_distance	Distance to parks/waterfront	3%	Logistic (saturates 10 min)
school_zone_rating	DOE quality score	3%	Borough-relative percentile
noise_composite	Traffic/nightlife/311	2%	Weighted composite
block_position	Mid-block vs corner	2%	Binary bonus/penalty
street_aesthetic	Tree-lined, landmark status	1%	Categorical bonus

Building & Amenities Features (15% total weight)

Feature	Description	Weight	Transform
elevator	Presence and type	2.5%	Binary/categorical
doorman	Full/part-time/virtual	2.0%	Categorical
gym	On-site fitness facility	1.5%	Binary + quality tier
roof_deck	Common roof access	1.5%	Binary + quality tier
maintenance_charges	Per sqft vs neighborhood	2.0%	Inverse percentile
building_size	Units and scale	2.0%	Categorical
laundry	In-building facilities	1.0%	Binary
pool	Swimming pool	1.0%	Binary
garage	Parking availability	1.0%	Binary
live_in_super	On-site superintendent	1.0%	Binary
storage	Unit storage available	0.5%	Binary

Unit & Layout Features (20% total weight)

Feature	Description	Weight	Transform
renovation_level	Estate to new development	7%	5-class categorical
washer_dryer	In-unit W/D	2%	Binary
dishwasher	Dishwasher present	1%	Binary
central_air	Central HVAC	1%	Binary
fireplace	Working fireplace	1%	Binary
layout_efficiency	Usable vs circulation space	3%	Percentage score
floor_height	Relative position	2%	Percentile in building
outdoor_space	Private balcony/terrace	3%	Sqrt(sqft) with cap

Bonuses & Penalties (±5-15% adjustment)

Feature	Type	Adjustment	Condition
ongoing_assessment	Penalty	-5%	Binary flag
pending_litigation	Penalty	-4%	Binary flag
estate_condition	Penalty	-3%	If not priced in
high_flip_tax	Penalty	-2%	>3% of sale
no_pets	Penalty	-2%	Binary flag
no_subletting	Penalty	-3%	Binary flag
tax_abatement	Bonus	+2%	Years remaining
sponsor_sale	Bonus	+1%	No board approval
low_carrying	Bonus	+3%	<20th percentile
river_views	Bonus	+4%	Text/photo verified
private_terrace	Bonus	+3%	>150 sqft

Appendix B: Neighborhood Taxonomy

Manhattan Segments

StreetEasy Neighborhood	Model Segment	Prestige Tier
Tribeca	Downtown Luxury	1
West Village	Downtown Premium	1
SoHo	Downtown Luxury	1
Upper East Side	Uptown Traditional	2
Upper West Side	Uptown Family	2
Chelsea	Midtown Premium	2
Gramercy Park	Midtown Traditional	1
East Village	Downtown Young	3
Lower East Side	Downtown Emerging	3
Financial District	Downtown Corporate	2
Midtown East	Midtown Corporate	2
Midtown West	Midtown Mixed	3
Harlem	Uptown Emerging	4
Washington Heights	Uptown Value	5
Inwood	Uptown Value	5

Brooklyn Segments

StreetEasy Neighborhood	Model Segment	Prestige Tier
DUMBO	Waterfront Luxury	1
Brooklyn Heights	Brownstone Premium	1
Park Slope	Brownstone Family	2
Williamsburg	Hipster Premium	2
Cobble Hill	Brownstone Traditional	2
Fort Greene	Cultural Premium	2
Prospect Heights	Central Family	3
Greenpoint	Hipster Emerging	3
Bushwick	Artist Emerging	4
Crown Heights	Central Value	4
Bed-Stuy	Central Emerging	4
Bay Ridge	Outer Traditional	4
Sunset Park	Outer Value	5

Property Type Classifications

- **Coop**: Cooperative ownership, board approval required
- **Condo**: Fee simple ownership, no board approval
- **Townhouse**: Single/multi-family, full building
- **Multifamily**: 2-4 unit investment property

Era Definitions

- **Prewar**: Built before 1940
- **Postwar**: Built 1940-1999
- **New Development**: Built 2000 or later

Appendix C: Mathematical Formulations

Logistic Transform Function

```
f(x) = 100 / (1 + e^(-k(x - μ)))
where:
  x = input value (e.g., distance in meters)
  μ = midpoint (inflection point)
  k = slope coefficient (steepness)

Example (Subway Distance):
  μ = 7 minutes (midpoint)
  k = 0.5 (moderate steepness)
  Results: 0 min → 98 pts, 7 min → 50 pts, 15 min → 2 pts
```

Price Gap to Score Mapping

```
score = 50 + 50 * tanh(price_gap / σ)
where:
  price_gap = (expected - asking) / expected
  σ = 0.08 (8% normalization factor)
  tanh = hyperbolic tangent (smooth S-curve)

Results:
  -20% gap (overpriced) → 15 points
  -8% gap → 35 points
  0% gap → 50 points
  +8% gap (underpriced) → 65 points
  +20% gap → 85 points
```

Diminishing Returns (Outdoor Space)

```
score = 100 * (1 - e^(-λ * sqrt(sqft)))
where:
  λ = 0.15 (decay parameter)
  sqft = outdoor space area

Results:
  0 sqft → 0 points
  100 sqft → 39 points
  400 sqft → 70 points
  900 sqft → 86 points
  1600 sqft → 92 points (diminishing returns)
```

Noise Composite Score

```
noise_score = Σ(wi * si) for i in [traffic, complaints, nightlife, emergency]

Traffic Score:
  st = 100 * (1 / (1 + e^(-0.1*(d - 40))))
  where d = distance to major road in meters

311 Complaint Score:
  sc = 100 * max(0, 1 - (n/200))
  where n = complaint count in 100m radius, 12 months

Nightlife Score:
  sn = 100 * max(0, 1 - sqrt(v/10))
  where v = venue count in 90m radius

Emergency Score:
  se = 100 / (1 + e^(-0.05*(d - 75)))
  where d = distance to hospital/fire in meters
```

Confidence Score Calculation

```
confidence = wc * coverage + wm * model_confidence + wd * data_quality

where:
  coverage = count(non_null_fields) / count(critical_fields)
  model_confidence = 1 - (prediction_interval_width / price)
  data_quality = weighted_avg(source_reliability)

  wc = 0.4 (coverage weight)
  wm = 0.4 (model weight)
  wd = 0.2 (data quality weight)
```

Segment-Relative Scoring

```
percentile_score = 100 * Φ((x - μs) / σs)

where:
  x = feature value
  μs = segment mean
  σs = segment standard deviation
  Φ = cumulative normal distribution

Applied per segment:
  segment = {borough, property_type, bed_bucket, era}
```

Appendix D: Sample API Responses

POST /api/v1/score

```
json
```

```
{
  "request": {
    "url": "https://streeteasy.com/building/163-east-81-street-new_york/3a",
    "manual_overrides": {
      "square_feet": 850
    }
  },
  "response": {
    "status": "success",
    "data": {
      "streetwise_score": 76,
      "confidence": 84,
      "interpretation": "Good Value",
      "verdict": {
        "assessment": "Fairly Priced",
        "confidence": "High",
        "expected_price": 1250000,
        "asking_price": 1195000,
        "price_gap_percentage": -4.4
      },
      "categories": {
        "fair_value": {
          "score": 71,
          "weight": 0.40,
          "drivers": [
            {
              "factor": "asking_below_expected",
              "impact": "+8",
              "description": "$55K below expected based on comps"
            },
            {
              "factor": "high_dom",
              "impact": "-5",
              "description": "67 days on market vs 45 avg"
            }
          ]
        },
        "location": {
          "score": 82,
          "weight": 0.20,
          "drivers": [
            {
              "factor": "subway_proximity",
```

```
      "impact": "+15",
      "description": "0.2 miles to 4/5/6 at 86th St"
    },
    {
      "factor": "school_zone",
      "impact": "+12",
      "description": "PS 6 (GreatSchools 9/10)"
    }
  ]
},
"building": {
  "score": 78,
  "weight": 0.15,
  "features": {
    "doorman": true,
    "elevator": true,
    "gym": false,
    "laundry": true,
    "amenity_score": 6.5
  }
},
"unit": {
  "score": 73,
  "weight": 0.20,
  "features": {
    "renovation": "recently_updated",
    "in_unit_laundry": false,
    "outdoor_space": null,
    "floor_height_percentile": 0.45
  }
},
"adjustments": {
  "bonuses": [
    {
      "type": "pets_allowed",
      "impact": "+2"
    }
  ],
  "penalties": [
    {
      "type": "high_maintenance",
      "impact": "-3",
      "detail": "$3.50/sqft vs $2.10 neighborhood avg"
    }
```

```
        ],
        "net_adjustment": -1
      }
    },
    "comparables": [
      {
        "address": "163 E 81st St #5B",
        "sale_date": "2024-10-15",
        "sale_price": 1175000,
        "price_per_sqft": 1382,
        "similarity_score": 0.94,
        "adjustments": {
          "floor": "+2%",
          "renovation": "-3%",
          "time": "+1.5%"
        }
      }
    ],
    "missing_data": [
      "light_quality",
      "floor_plan"
    ],
    "data_sources": {
      "listing": "StreetEasy",
      "building": "PLUTO",
      "sales": "ACRIS",
      "schools": "NYC DOE"
    }
  }
 }
}
```


GET /api/v1/market

```json
```

```json
{
  "request": {
    "neighborhood": "Upper East Side",
    "timeframe": "12_months"
  },
  "response": {
    "median_prices": {
      "current": 1425000,
      "year_ago": 1350000,
      "change_percentage": 5.6
    },
    "price_per_sqft": {
      "current": 1456,
      "year_ago": 1398,
      "trend": "increasing"
    },
    "inventory": {
      "current_listings": 342,
      "average_dom": 52,
      "absorption_rate": 0.34,
      "months_supply": 2.9
    },
    "segments": {
      "1_bedroom": {
        "median_price": 895000,
        "inventory": 128
      },
      "2_bedroom": {
        "median_price": 1750000,
        "inventory": 156
      },
      "3_plus_bedroom": {
        "median_price": 3450000,
        "inventory": 58
      }
    }
  }
}
```

Document Version History

- v2.0: Complete technical specification (December 2024)
- v1.0: Initial draft specification

Contact

For questions about this specification or the Streetwise platform, contact: [project contact information]