MLS Data Extraction Slash Command - Implementation Plan

Date: 2025-11-05 **Branch**: feature/enhance-valuation-variables **Purpose**: Extract enhanced MLS field set to structured CSV/Excel format for analysis

---

EXECUTIVE SUMMARY

Create a new slash command `/Financial_Analysis:extract-mls` that extracts all 24 valuation variables plus broker comments and metadata from MLS PDF reports into a structured spreadsheet format (CSV or Excel).

Key Benefits: - Streamlines data entry for comparative analysis - Captures all enhanced variables (bay depth, lot size, HVAC, sprinklers, etc.) - Preserves broker comments for qualitative insights - Enables bulk import into valuation calculator - Professional Excel formatting for client delivery

---

SCOPE

In Scope

- Extract all 24 valuation variables (10 core + 14 optional)
- Extract broker comments/remarks fields
- Support both CSV (simple) and Excel (formatted) output
- Handle multiple properties from single PDF (batch extraction)
- Robust parsing with fallback for missing fields
- Eastern Time timestamps for output filenames
- Save to `Reports/` folder with standard naming convention

Out of Scope (Phase 2)

- Automatic distance calculation (use existing `/relative-valuation` workflow)
- Data validation/quality checks (assume clean MLS data)
- Integration with CRM systems
- Web scraping from MLS websites
- Image extraction (photos, floor plans)

INPUT SPECIFICATION

Command Syntax

```
        /Financial_Analysis:extract-mls <pdf-path> [--format=csv|excel] [--
subject=<address>]
```

Parameters: - `<pdf-path>` (required): Path to MLS PDF report - `--format` (optional): Output format, default `excel` - `csv`: Plain CSV file - `excel`: Formatted XLSX with headers, filters, column sizing - `--subject` (optional): Address of subject property to mark with `is_subject=true`

Example:

```
      /Financial_Analysis:extract-mls /path/to/Mississauga_100-400k_sf_for_lease.pdf --format=excel --subject="2550 Stanfield"
```

OUTPUT SPECIFICATION

File Naming Convention

Format: `YYYY-MM-DD_HHMMSS_mls_extraction_<market>.{csv|xlsx}` (Eastern Time)

Example: `2025-11-05_183047_mls_extraction_mississauga.csv`

Output Location

`Reports/` folder (consistent with other financial analysis outputs)

FIELD MAPPING

Core Variables (10 fields)

Column Name	Data Type	MLS Source Field	Example	Notes
`address`	String	Address	"2550 Stanfield Rd, Mississauga, ON L4Y 1S2, Canada"	Complete geocodable format
`unit`	String	Unit/Suite	"Opt 2"	Separate from address
`available_sf`	Integer	Area	186559	Rentable square footage
`net_asking_rent`	Float	$/SF Net or Rent	13.95	Net asking rent per SF
`tmi`	Float	Addl Rent or T.M.I.	3.01	Additional rent/TMI per SF
`year_built`	Integer	Yr Blt	2020	Year constructed
`clear_height_ft`	Float	Clear Ht	34.0	Clear height in feet

`pct_office_space`	Float	% Office	0.03	Percentage (0.03 = 3%)
`parking_ratio`	Float	Parking/1000	1.0	Spaces per 1,000 SF
`class`	Integer	Class	2	A=1, B=2, C=3

Existing Optional Variables (6 fields)

Column Name	Data Type	MLS Source Field	Example	Notes
`shipping_doors_tl`	Integer	Truck Level or Ship (TL)	16	Truck-level doors
`shipping_doors_di`	Integer	Drive-In or Ship (DI)	3	Drive-in doors
`power_amps`	Integer	Power	3000	Electrical service in amps
`trailer_parking`	Boolean	Trailer Pkg or Outside Pkg	true	Trailer parking available
`secure_shipping`	Boolean	Secure Ship	false	Secure/enclosed shipping
`excess_land`	Boolean	Excess Land	false	Additional developable land

New Optional Variables (8 fields)

Column Name	Data Type	MLS Source Field	Example	Extraction Logic
`bay_depth_ft`	Float	Bay Size	55.0	Parse “55 x 52” → 55.0 (regex)
`lot_size_acres`	Float	Lot Irreg or Lot Size Area	11.112	Parse acres or convert sq ft
`hvac_coverage`	Integer	A/C	1	Y=1, Part=2, N=3 (ordinal)
`sprinkler_type`	Integer	Sprinklers + Client Remks	1	ESFR=1, Standard=2, None=3
`building_age_years`	Integer	Calculated	5	report_year - year_built
`rail_access`	Boolean	Rail	false	Y/N boolean

`crane`	Boolean	Crane	false	Y/N boolean
`occupancy_status`	Integer	Occup	1	Vacant=1, Tenant=2 (ordinal)

Metadata & Comments (9 fields)

Column Name	Data Type	MLS Source Field	Example	N
`availability_date`	String	Avail	“Immediate”	Dat “Im
`days_on_market`	Integer	DOM	119	Day mar
`mls_number`	String	ML# or MLS#	“C1234567”	MLS num
`broker_name`	String	List Off or Agent	“CBRE Limited”	List brol
`client_remarks`	Text	Client Remks	“ESFR sprinklers, new LED…”	Mar desc
`is_subject`	Boolean	Derived	true	Mat agai subj para
`reported_market`	String	Derived	“Mississauga, ON”	Mar nam infe fron
`report_generated_at`	String (ISO 8601)	Derived	“2025-11-05T18:30:47-05:00”	Con run time (ET)
`source_pdf`	String	Derived	“Mississauga_100-400k_sf_for_lease.pdf”	Orig PDF filen

Total Columns: 33 (10 core + 6 existing optional + 8 new optional + 9 metadata)

Note: The 3 report-level metadata fields (`reported_market`, `report_generated_at`, `source_pdf`) are stored in the JSON `report_metadata` object but duplicated as columns in each CSV/Excel row for convenience.

TECHNICAL APPROACH

Workflow Architecture

Claude Code Extraction Approach (similar to `/relative-valuation`):

```
PDF → [Claude Code Extraction] → JSON → [Python Formatter] →
```

```
Excel/CSV
```

Why Claude Code for Extraction?

Advantages over Python PDF parsing libraries (pdfplumber/markitdown):

1. **Robust to format variations**
   - Different MLS providers use different layouts (CBRE, JLL, Cushman, Colliers)
   - Field names vary: “Addl Rent” vs “T.M.I.” vs “OpEx”
   - LLM understands context and adapts automatically
2. **Intelligent parsing**
   - Interprets ambiguous data: “Part A/C” → partial HVAC coverage
   - Detects ESFR mentions in free-text “Client Remarks”
   - Converts units intelligently: “484,280 Sq Ft” → 11.112 acres
3. **Better error recovery**
   - Gracefully handles missing fields
   - Can flag uncertain extractions
   - Works with scanned PDFs and complex layouts
4. **No brittle regex/parsing code**
   - No maintenance burden when MLS formats change
   - Faster implementation (no complex parsing logic)
   - Higher quality output

Technology Stack

Phase 1: Claude Code Extraction (Slash Command) - Claude Code reads PDF using Read tool - LLM extracts all 33 fields with contextual understanding - Outputs structured JSON to `Reports/` folder - **Dependencies**: None (built into Claude Code)

Phase 2: Python Excel Formatter - Python `openpyxl` library for professional Excel formatting - Takes JSON input from Phase 1 - Applies formatting: - Header row: bold white text on dark blue background, frozen - Auto-filter enabled on all columns - Column width auto-sizing - Number formatting (currency for rent/TMI, percentages, integers) - Conditional formatting for `is_subject` row (yellow highlight) - Outputs CSV (simple) or XLSX (formatted) - **Dependencies**: `openpyxl` only

Extraction Strategy

Step 1: Slash Command Invocation

```
        /Financial_Analysis:extract-mls /path/to/mls_report.pdf --
format=excel --subject="2550 Stanfield"
```

Step 2: Claude Code Extraction

```
        1. Read PDF using Read tool
        2. Identify all property listings in the document
        3. For each property, extract all 33 fields:
           - Core variables (10): address, unit, SF, rent, TMI, year built,
clear height, % office, parking ratio, class
           - Existing optional (6): shipping doors (TL/DI), power, trailer
parking, secure shipping, excess land
           - New optional (8): bay depth, lot size, HVAC, sprinklers,
building age, rail, crane, occupancy
```

```
           - Metadata (9): availability date, DOM, MLS#, broker, comments,
is_subject, market, timestamp, source PDF
        4. Apply intelligent parsing:
           - Bay depth: Extract from "55 x 52" → 55.0
           - Lot size: Convert "484,280 Sq Ft" → 11.112 acres
           - HVAC: Map "Y"→1, "Part"→2, "N"→3
           - Sprinklers: Check "Client Remks" for "ESFR" mention → 1
           - Building age: Calculate from report year
        5. Mark subject property (fuzzy match against --subject parameter)
        6. Write JSON to Reports/ with timestamp
```

Step 3: Python Formatting (automatic, called by slash command)

```python
import json
from openpyxl import Workbook

# Load JSON
with open(json_path) as f:
    data = json.load(f)

# Format to Excel or CSV based on --format flag
if format == 'excel':
    create_formatted_excel(data, output_path)
else:
    create_csv(data, output_path)
```

IMPLEMENTATION STEPS

Phase 1: Command Setup (2 hours)

1. Create `.claude/commands/Financial_Analysis/extract-mls.md` slash command
2. Define command workflow:
   - **Step 1**: Read PDF using Read tool
   - **Step 2**: Extract all 33 fields using Claude Code (LLM extraction)
   - **Step 3**: Create JSON output with all properties
   - **Step 4**: Call Python formatter to generate Excel/CSV
   - **Step 5**: Save outputs to Reports/ folder
3. Document extraction requirements and field mapping
4. Add command to README

Phase 2: Claude Code Extraction Logic (4 hours)

5. Implement extraction prompt in slash command:
   - Provide field mapping table (33 fields)
   - Specify parsing rules:
     - Bay depth: Parse first number from “Bay Size” (e.g., “55 x 52” → 55.0)
     - Lot size: Extract acres or convert sq ft to acres (÷ 43,560)
     - HVAC coverage: Map Y=1, Part=2, N=3
     - Sprinkler type: Check “Client Remks” for ESFR → 1, else Standard=2, None=3
     - Building age: Calculate from report year (derived from report_generated_at) minus year_built
     - Complete addresses: Format as “Street, City, ON PostalCode, Canada”

    - Subject property matching: Fuzzy match against –subject parameter
    - Error handling: Use NULL for missing fields, continue processing
6. Create JSON schema template:

```
{
  "report_metadata": {
    "analysis_date": "2025-11-05",
    "market": "Mississauga - Industrial",
    "report_generated_at": "2025-11-05T18:30:47-05:00",
    "subject_property": "2550 Stanfield Rd",
    "source_pdf": "Mississauga_100-400k_sf_for_lease.pdf"
  },
  "properties": [
    {
      "address": "2550 Stanfield Rd, Mississauga, ON L4Y 1S2,
Canada",
      "unit": "Opt 2",
      "available_sf": 186559,
      "net_asking_rent": 13.95,
      "tmi": 3.01,
      "year_built": 2020,
      "clear_height_ft": 34.0,
      "pct_office_space": 0.03,
      "parking_ratio": 1.0,
      "class": 2,
      "shipping_doors_tl": 16,
      "shipping_doors_di": 3,
      "power_amps": 3000,
      "trailer_parking": false,
      "secure_shipping": false,
      "excess_land": false,
      "bay_depth_ft": null,
      "lot_size_acres": null,
      "hvac_coverage": 2,
      "sprinkler_type": 2,
      "building_age_years": 5,
      "rail_access": false,
      "crane": false,
      "occupancy_status": 1,
      "availability_date": "Immediate",
      "days_on_market": 45,
      "mls_number": "C1234567",
      "broker_name": "CBRE Limited",
      "client_remarks": "Partial A/C, standard sprinklers, excellent
access to 401/407",
      "is_subject": true,
      "reported_market": "Mississauga, ON",
      "report_generated_at": "2025-11-05T18:30:47-05:00",
      "source_pdf": "Mississauga_100-400k_sf_for_lease.pdf"
    }
  ]
}
```

Note: The 3 report-level metadata fields are stored at the top level in `report_metadata` AND duplicated in each property object for CSV/Excel export convenience.

Phase 3: Python Output Formatting (3 hours)

7. Implement CSV writer

8. Implement Excel writer with formatting:

```
        from openpyxl import Workbook
        from openpyxl.styles import Font, PatternFill, Alignment
        from openpyxl.utils import get_column_letter

        # Create workbook
        wb = Workbook()
        ws = wb.active
        ws.title = "MLS Extraction"

        # Write headers
        headers = ['address', 'unit', 'available_sf', ...]
        ws.append(headers)

        # Format header row
        header_fill = PatternFill(start_color='366092',
end_color='366092', fill_type='solid')
        header_font = Font(bold=True, color='FFFFFF')
        for cell in ws[1]:
            cell.fill = header_fill
            cell.font = header_font

        # Freeze header row
        ws.freeze_panes = 'A2'

        # Enable auto-filter
        ws.auto_filter.ref = ws.dimensions

        # Auto-size columns
        for column in ws.columns:
            max_length = max(len(str(cell.value or '')) for cell in
column)

ws.column_dimensions[get_column_letter(column[0].column)].width =
max_length + 2
```

9. Add conditional formatting for subject property row:

```
        from openpyxl.formatting.rule import Rule
        from openpyxl.styles.differential import DifferentialStyle

        # Highlight subject property in yellow
        yellow_fill = PatternFill(start_color='FFFF00',
end_color='FFFF00', fill_type='solid')
        # Apply to rows where is_subject = TRUE
```

Phase 4: Integration & Testing (3 hours)

10. Create `MLS_Extractor/format_mls.py` Python script
11. Integrate script call into slash command workflow
12. Test end-to-end with Mississauga dataset (23 properties):
    - Claude Code extracts to JSON
    - Python formatter generates Excel/CSV
    - Validate all 33 fields extracted correctly
    - Verify subject property marked correctly
13. Test both CSV and Excel outputs
14. Validate Excel formatting (headers, filters, highlighting)

15. Cross-platform testing (Microsoft Excel, Google Sheets, LibreOffice)

Phase 5: Documentation (2 hours)

16. Complete slash command documentation with examples
17. Create `MLS_Extractor/README.md` with usage guide
18. Document field mapping table in `MLS_Extractor/FIELD_MAPPING.md`
19. Add troubleshooting guide (common issues, solutions)
20. Update `.claude/commands/README.md` to include `/Financial_Analysis:extract-mls`
21. Create example output files for reference

EXAMPLE OUTPUT

CSV Format (first 3 rows)

```
address,unit,available_sf,net_asking_rent,tmi,year_built,clear_heigh

"2550 Stanfield Rd, Mississauga, ON L4Y 1S2, Canada",Opt
2,186559,13.95,3.01,2020,34.0,0.03,1.0,2,16,3,3000,False,False,False,
 Limited,"Partial A/C, standard sprinklers, excellent access to
401/407",True,"Mississauga, ON",2025-11-05T18:30:47-
05:00,Mississauga_100-400k_sf_for_lease.pdf
"795 Hazelhurst Rd, Mississauga, ON L5J 2Z6,
Canada",,215124,1.00,4.00,2021,36.0,0.05,1.2,1,34,2,2000,False,False,
 2026,89,C7654321,JLL,"ESFR sprinklers, full A/C, deep 55' bays
ideal for racking",False,"Mississauga, ON",2025-11-05T18:30:47-
05:00,Mississauga_100-400k_sf_for_lease.pdf
"560 Slate Dr, Mississauga, ON L5T 0A1,
Canada",,160485,1.00,0.00,2019,40.0,0.02,1.5,1,26,2,,True,False,False
 2025,119,C9876543,Cushman & Wakefield,"ESFR, 40' clear, trailer
parking, large lot",False,"Mississauga, ON",2025-11-05T18:30:47-
05:00,Mississauga_100-400k_sf_for_lease.pdf
```

Excel Format

Sheet Name: MLS Extraction

Header Row (Row 1): Bold white text on dark blue background, frozen **Filters**: Enabled on all columns **Column Widths**: Auto-sized to content **Subject Row**: Yellow highlight background **Number Formatting**: - `available_sf`: `#,##0` (no decimals) - `net_asking_rent`, `tmi`: `$#,##0.00` - `clear_height_ft`, `bay_depth_ft`: `#,##0.0` - `pct_office_space`: `0.0%` - `parking_ratio`, `lot_size_acres`: `#,##0.00`

Conditional Formatting: - `is_subject = TRUE`: Entire row highlighted in yellow

---

VALIDATION PLAN

Test Case 1: Mississauga Dataset (23 properties)

Input: `skillsdevdocs/Mississauga_100-400k_sf_for_lease.pdf` **Expected**: 23 rows extracted, all 33 fields populated **Validation**: - All addresses in correct geocodable format - Bay depth parsed for properties with "Bay Size" field - Lot sizes converted to acres (including sq ft → acre conversion) - ESFR sprinklers detected from Client Remarks - Subject property (2550 Stanfield Opt 2) marked with `is_subject=true`

Test Case 2: Missing Data Handling

Input: Property with incomplete MLS data **Expected**: Empty cells or NULL values, no crash **Validation**: - Missing bay size → `bay_depth_ft` = NULL - Missing lot size → `lot_size_acres` = NULL - Missing A/C → `hvac_coverage` = 3 (default to N)

Test Case 3: CSV vs Excel Output

Input: Same PDF, generate both formats **Expected**: Identical data, different formatting **Validation**: - CSV: plain text, no formatting - Excel: formatted headers, filters, column sizing, subject highlight

Test Case 4: Subject Property Matching

Input: `--subject="2550 Stanfield"` **Expected**: Only rows with "2550 Stanfield" in address get `is_subject=true` **Validation**: - Fuzzy matching (partial address match) - Case-insensitive - Multiple units at same address handled correctly

---

ERROR HANDLING

Scenario 1: PDF Parsing Failure

Error: PDF has unusual structure, tables not detected **Handling**: Log error, fall back to manual extraction prompt for user

Scenario 2: Missing Required Fields

Error: Address or Available SF missing **Handling**: Skip row, log warning with property identifier

Scenario 3: Invalid Data Types

Error: Clear height = "N/A" (expected float) **Handling**: Set to NULL, log warning, continue processing

Scenario 4: Output File Already Exists

Error: Timestamp collision (same second) **Handling**: Append counter suffix: `_mls_extraction_mississauga_1.xlsx`

INTEGRATION WITH EXISTING WORKFLOW

Current Workflow

1. User runs `/relative-valuation` command
2. Command extracts data and creates JSON input file
3. JSON fed to Python calculator
4. Calculator generates report

Enhanced Workflow with `/Financial_Analysis:extract-mls`

1. User runs `/Financial_Analysis:extract-mls` to create Excel spreadsheet
2. User reviews/edits spreadsheet (manual QA step)
3. User converts Excel to JSON (new utility script or manual)
4. User runs `/relative-valuation` with JSON input (skip PDF extraction)
5. Calculator generates report

Alternative: Direct integration 1. `/Financial_Analysis:extract-mls` generates both Excel (for review) AND JSON (for automation) 2. JSON automatically fed to distance calculator 3. User can run `/relative-valuation` immediately or review Excel first

---

DEPENDENCIES

Python Libraries

- `openpyxl` - Excel file creation and formatting (only external dependency)
- Standard library: `csv`, `json`, `datetime`, `zoneinfo`

Installation

```
pip install openpyxl
```

Note: No PDF parsing libraries required! Claude Code handles all PDF extraction using the Read tool and LLM understanding.

FUTURE ENHANCEMENTS (Phase 2)

1. Data Validation Rules

- Flag properties with missing critical fields (address, SF, rent)
- Highlight outliers (rent >2x median, clear height <20 ft)
- Validate postal codes, phone numbers

2. Excel Charts & Pivot Tables

- Rent distribution histogram
- Clear height vs area scatter plot

- Property class breakdown pie chart
- Pre-configured pivot table for quick analysis

3. Multi-Sheet Workbooks

- Sheet 1: Raw data extraction
- Sheet 2: Summary statistics
- Sheet 3: Data quality report (missing fields, outliers)

4. CRM/MLS Integration

- Direct API integration with MLS systems (CREA, TREB)
- Automatic updates when listings change
- Historical price tracking

5. Batch Processing

- Process multiple PDFs in single command
- Aggregate into master spreadsheet
- Cross-market comparison (Mississauga vs Brampton vs Vaughan)

SUCCESS CRITERIA

Functional Requirements

- □ Extract all 33 fields from MLS PDF
- □ Support CSV and Excel output formats
- □ Handle 20+ properties per PDF
- □ Robust parsing with 95%+ field accuracy
- □ Subject property identification
- □ Professional Excel formatting

Performance Requirements

- □ Process 25-property PDF in <30 seconds
- □ File size <5 MB for 100 properties

Quality Requirements

- □ Zero crashes on malformed PDFs (graceful degradation)
- □ Field accuracy validated against manual extraction (spot check 10%)
- □ Excel files open correctly in Microsoft Excel and Google Sheets

---

TIMELINE ESTIMATE

Phase	Tasks	Estimated Effort
Phase 1	Command setup, workflow definition	2 hours
Phase	Claude Code extraction prompt & JSON	

2	schema	4 hours
Phase 3	Python formatting (CSV + Excel)	3 hours
Phase 4	Integration, end-to-end testing	3 hours
Phase 5	Documentation, examples	2 hours
Total		**14 hours**

Reduction from original 20-hour estimate: No complex PDF parsing code to write, test, or maintain. Claude Code handles extraction with LLM intelligence.

RISKS & MITIGATION

Risk 1: LLM Extraction Accuracy

Impact: Medium - Claude Code might misinterpret ambiguous fields **Mitigation**: - Provide detailed field mapping table in extraction prompt - Include examples for each field type - Test with diverse MLS formats (CBRE, JLL, Cushman, Colliers) - Validate against manual extraction (10% spot check) - User reviews Excel output before using for analysis

Advantage over Python parsing: LLM adapts to format variations automatically, whereas regex/parsing code breaks on unexpected formats.

Risk 2: API Costs & Latency

Impact: Low - LLM extraction requires API calls **Mitigation**: - Typical MLS PDF (23 properties) = ~50K tokens = ~$0.15 per extraction - Latency: ~30-60 seconds for 23 properties (acceptable for batch workflow) - Much cheaper than manual data entry ($50-100/hour labor cost)

Trade-off: Higher per-extraction cost, but lower maintenance cost (no brittle parsing code to fix when MLS formats change).

Risk 3: Excel Library Compatibility

Impact: Low - openpyxl may not support all Excel features **Mitigation**: - Test in Microsoft Excel, LibreOffice Calc, Google Sheets - Stick to basic formatting (fonts, colors, filters) - Avoid advanced features (macros, complex formulas)

DELIVERABLES

Code

1. `.claude/commands/Financial_Analysis/extract-mls.md` - Slash

command with Claude Code extraction workflow
2. `MLS_Extractor/format_mls.py` - Python formatter for CSV/Excel output
3. `MLS_Extractor/json_schema.json` - JSON schema template for extraction

Documentation

4. `MLS_Extractor/README.md` - Usage guide and examples
5. `MLS_Extractor/FIELD_MAPPING.md` - Complete field reference (33 fields)
6. Updated `.claude/commands/README.md` - Add `/Financial_Analysis:extract-mls` to command list
7. Example output files in `Reports/` folder:
   - `*_mls_extraction.json` - Raw JSON from Claude extraction
   - `*_mls_extraction.csv` - CSV format
   - `*_mls_extraction.xlsx` - Formatted Excel

Testing & Validation

8. End-to-end test with Mississauga dataset (23 properties)
9. Validation report comparing Claude extraction vs manual extraction (10% spot check)
10. Cross-platform Excel compatibility test results

NEXT STEPS

1. □ Review and approve this implementation plan
2. □ Create GitHub issue #11 to track development
3. Install dependencies: `pip install openpyxl`
4. Implement Phase 1 (slash command setup with extraction workflow)
5. Implement Phase 2 (Claude Code extraction prompt with field mapping)
6. Implement Phase 3 (Python Excel/CSV formatter)
7. Test end-to-end with Mississauga PDF (23 properties)
8. Validate extraction accuracy (spot check 10%)
9. Complete documentation and examples
10. Merge to main branch

END OF PLAN