

---

# MCP-PowerBI-Finvision

AI-Powered Power BI Analysis Server

Version 7.7+ | User Guide

*"Unlock the full potential of your Power BI models with AI assistance"*

50+ Tools

DAX Intelligence

Best Practices

Auto Docs

**What's Inside:** Learn how to leverage AI to analyze Power BI models, write DAX measures, debug visuals, optimize performance, and generate documentation - all through natural language conversations.

# Table of Contents

---

1	<b>Why Claude?</b>	AI capabilities, connectors & strengths
2	<b>What is MCP?</b>	Understanding the Model Context Protocol
3	<b>What is MCP-PowerBI-Finvision?</b>	Your AI-powered Power BI companion
4	<b>Getting Started</b>	Installation and first connection
5	<b>Tools Overview</b>	50+ tools across 10 categories
6	<b>Complete Tool Reference</b>	Detailed tool documentation
7	<b>Common Workflows</b>	Step-by-step guides
8	<b>Tips &amp; Best Practices</b>	Get the most out of MCP
9	<b>Beyond Power BI</b>	VS Code, Fabric, PySpark & more
10	<b>The Future: Fabric MCP</b>	What's coming next

# 1. Why Claude?

INFO

**Claude** is Anthropic's most capable AI assistant, built on cutting-edge research in AI safety and helpfulness. It combines deep technical knowledge with nuanced understanding, making it an ideal partner for complex analytical and development tasks across any domain.

## Claude's Key Strengths

<b>Advanced Reasoning</b> Breaks down complex problems into logical steps, handles multi-step analysis, and provides clear explanations of its thinking	<b>Extended Context</b> Processes up to 200K tokens - entire codebases, long documents, or complex datasets in a single conversation
<b>Code Excellence</b> Writes production-quality code in 20+ languages including Python, TypeScript, SQL, DAX, and more with best practices built-in	<b>Learning Partner</b> Explains concepts at any level, teaches patterns, and helps you grow your skills while solving real problems

## What Claude Can Do

- **Software Development** - Write, debug, refactor, and review code across any language or framework
- **Data Analysis** - Analyze datasets, write queries, create visualizations, and extract insights
- **Technical Writing** - Generate documentation, API specs, architecture diagrams, and technical guides
- **Problem Solving** - Debug issues, optimize performance, design systems, and plan implementations
- **Research & Learning** - Explain complex topics, summarize papers, and answer technical questions

## Ways to Use Claude

Interface	Best For	Key Features
Claude.ai	General tasks, conversations	Web interface, file uploads, artifacts, projects
Claude Desktop	Power users, MCP integrations	Native app, MCP server support, local tools
Claude Code (CLI)	Developers, automation	Terminal-based, git integration, agentic coding
API	Applications, workflows	Programmatic access, batch processing, custom apps
IDE Extensions	In-editor assistance	VS Code, JetBrains - code completion & chat

## Extensibility with MCP

Claude's capabilities can be extended through the **Model Context Protocol (MCP)** - an open standard that allows Claude to connect to external tools and data sources. This guide focuses on one such extension: **MCP-PowerBI-Finvision**, which gives Claude deep access to Power BI Desktop.

### TIP

With MCP, Claude goes from being a helpful assistant to an **active collaborator** that can read your files, query your databases, interact with your tools, and execute operations on your behalf - all while maintaining the safety and helpfulness that Claude is known for.

## 2. What is MCP?

### INFO

**Model Context Protocol (MCP)** is like a universal translator between AI assistants (like Claude) and software tools. It allows AI to understand and interact with specialized applications in a standardized way.

### Think of it Like This...

Imagine you have a brilliant assistant who speaks English, but your Power BI files speak a completely different "language." MCP acts as an interpreter:

#### The Communication Flow:



- **Translates requests** from natural language into technical commands
- **Executes operations** on your behalf (create measures, analyze performance)
- **Returns results** in a way you can understand

### A Real Example

**You:** "Show me all measures in my Sales table"

**Claude:** Understands your intent, calls 02\_Measure\_Operations with table='Sales'

**Result:** A nicely formatted list of all 15 measures with descriptions and DAX expressions

### TIP

You don't need to know DAX, M queries, or technical Power BI commands. Just describe what you want in plain English!

### 3. What is MCP-PowerBI-Finvision?

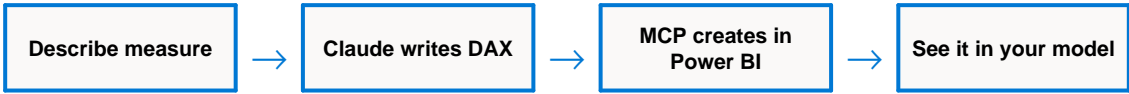
INFO

**MCP-PowerBI-Finvision** is a specialized MCP server designed specifically for Power BI Desktop. It provides **50+ tools** across **10 categories** that enable AI assistants to analyze, modify, and optimize your Power BI models.

#### Key Capabilities

<b>Model Analysis</b> Examine tables, columns, measures	<b>DAX Intelligence</b> Analyze, debug, optimize DAX
<b>Best Practices</b> 120+ rules to identify issues	<b>Performance</b> Find slow queries & optimize
<b>CRUD Operations</b> Create, update, delete objects	<b>Visual Debugging</b> Analyze visuals & filter contexts
<b>Documentation</b> Auto-generate Word documents	<b>Offline Analysis</b> Analyze PBIP without Power BI

#### Example: Creating a Measure



**You:** "Create a measure for year-over-year sales growth"

**Claude:** Writes the DAX and creates the measure for you

**Result:** YoY Growth = DIVIDE([Current Sales] - [Prior Year Sales], [Prior Year Sales])

## 4. Getting Started

---

### Prerequisites

- **Windows 10/11** (64-bit)
- **Power BI Desktop** installed
- **Claude Desktop** or MCP-compatible AI client

### Installation

Quick Install:



The **setup.bat** script automatically configures Claude Desktop to use the MCP server. It detects your Python installation, creates the virtual environment, installs dependencies, and updates your Claude configuration file.

### Your First Connection

#### NOTE

- Step 1:** Open Power BI Desktop with your .pbix file
- Step 2:** Open Claude Desktop
- Step 3:** Type: "Connect to my Power BI model"
- Step 4:** Claude automatically detects and connects!
- Step 5:** Start exploring: "Give me an overview"

## 5. Tools Overview

The server provides **50+ tools** organized into **10 categories**. You don't need to memorize them - just describe what you want!

Category	Purpose	#
01 Connection	Detect and connect to Power BI Desktop	2
02 Model Ops	CRUD for tables, columns, measures, relationships	8
03 Batch/Trans	Execute multiple operations atomically	2
04 Query/Search	Run DAX queries and search objects	6
05 DAX Intelligence	Analyze, debug, and optimize DAX	5
06 Analysis	Model analysis and best practices	3
07 PBIP Analysis	Offline project file analysis	10
08 Documentation	Generate model documentation	2
09 Debug	Visual debugging and advanced analysis	11
10 Help	User guides and references	1

### Natural Language to Tools

You Say...	Tool Used
"List all measures"	02_Measure_Operations (list)
"Show DAX for Total Sales"	02_Measure_Operations (get)
"Create a YTD measure"	02_Measure_Operations (create)
"Find measures using CALCULATE"	04_Search_String
"Analyze this measure"	05_DAX_Intelligence
"Check for best practice issues"	06_Full_Analysis
"Generate documentation"	08_Generate_Documentation_Word



# 6. Complete Tool Reference

## 01 Connection Tools

Detect and connect to Power BI Desktop instances

Connection Flow:



### 01\_Detect\_PBI\_Instances

Scans your system for running Power BI Desktop instances. Returns file names and ports.

**You:** "What Power BI files do I have open?"

**Claude:** Scans for running instances

**Result:** Found 2: Sales\_Model.pbix (port 52000), Finance\_Model.pbix (port 52100)

### 01\_Connect\_To\_Instance

Establishes connection to a Power BI instance. Auto-detects if only one is running.

**You:** "Connect to my Power BI model"

**Claude:** Auto-detects and connects

**Result:** Connected to Sales\_Model.pbix - 15 tables, 42 measures

## 02 Model Operations

Complete CRUD for all model objects

### 02\_Table\_Operations

Manage tables: list, describe, preview data, create, rename, delete.

list	Get all tables
describe	Table details with columns/measures
preview	Sample data rows
create/rename/delete	Modify tables

**You:** "Show me what's in the Sales table"

**Claude:** Gets structure and sample data

**Result:** Sales: 8 columns, 5 measures, 2 relationships. Shows first 10 rows.

## 02\_Measure\_Operations

Create, read, update, delete measures. The most frequently used tool!

<code>list</code>	List measure names (fast)
<code>get</code>	Get measure WITH DAX expression
<code>create</code>	Create new measure
<code>update/delete/rename</code>	Modify measures

**You:** "Show DAX for Profit Margin"

**Claude:** Retrieves measure

**Result:** `DIVIDE([Revenue] - [Cost], [Revenue])`

**You:** "Create average order value measure"

**Claude:** Creates measure

**Result:** Created: Avg Order Value = `DIVIDE([Total Sales], [Order Count])`

## 02\_Column\_Operations

Work with columns: statistics, distributions, create calculated columns.

<code>list/get</code>	View columns and details
<code>statistics</code>	Min, max, distinct count, blanks
<code>distribution</code>	Value frequency analysis
<code>create</code>	Create calculated column

## 02\_Relationship\_Operations

Manage relationships between tables.

<code>list</code>	All relationships
<code>find</code>	Relationships for a table
<code>create/delete</code>	Modify relationships

## 02\_Calculation\_Group\_Operations

Manage calculation groups for time intelligence.

*"List calculation groups"*

*"Create Time Intelligence group"*

## **02\_Role\_Operations**

View Row-Level Security (RLS) and Object-Level Security (OLS) roles.

## **02\_TMDL\_Operations**

TMDL automation: export, find/replace, bulk rename with reference updates.

## 03 Batch & Transactions

Execute multiple operations efficiently

### Batch Flow:



### 03\_Batch\_Operations

Execute multiple operations in one batch. 3-5x faster than individual calls!

**You:** "Create 5 measures: Total Sales, Avg Sales, YTD, MTD, Count"

**Claude:** Uses batch operation

**Result:** Created 5 measures in 1.2s (vs 6+ seconds individually)

### 03\_Manage\_Transactions

Transaction control: begin, commit, rollback for complex operations.

## 04 Query & Search

Execute DAX queries and search objects

### 04\_Run\_DAX

Execute DAX queries with automatic safety limits.

<code>query</code>	The DAX query (EVALUATE statement)
<code>top_n</code>	Max rows (default: 100)
<code>mode</code>	'auto', 'analyze', 'profile', 'simple'

**You:** "Show me sales by category"

**Claude:** Runs SUMMARIZECOLUMNS query

**Result:** Electronics: \$1.2M, Clothing: \$800K, Home: \$600K...

### 04\_Search\_Objects

Search across tables, columns, and measures by name.

**You:** "Find everything related to 'revenue'"

**Claude:** Searches all objects

**Result:** Found: Revenue table, Total Revenue measure, Revenue\_Category column

## 04\_Search\_String

Search inside measure DAX expressions for patterns.

**You:** "Which measures use CALCULATE?"

**Claude:** Searches DAX expressions

**Result:** 15 measures found: YTD Sales, MTD Sales, Filtered Revenue...

## 04\_Get\_Data\_Sources / 04\_Get\_M\_Expressions / 04\_Validate\_DAX

List data sources, view M expressions, validate DAX syntax before execution.

## 05 DAX Intelligence

Your AI-powered DAX expert!

### DAX Analysis Flow:



### 05\_DAX\_Intelligence

The most powerful DAX tool. Analyzes context transitions, detects anti-patterns, provides step-by-step debugging with optimization suggestions.

<code>expression</code>	DAX or measure name (fuzzy match!)
<code>analyze</code>	Context transition analysis
<code>debug</code>	Step-by-step debugging
<code>report</code>	Comprehensive 8-section report

**You:** "What's wrong with my Profit Margin measure?"

**Claude:** Runs analysis

**Result:** Found: unnecessary CALCULATE, missing blank handling. Fix provided.

**You:** "Debug CALCULATE(SUM(Sales[Amount]), FILTER(ALL(Date)...))"

**Claude:** Steps through filter context

**Result:** Step 1: ALL removes filters... Step 2: FILTER applies Year=2024...

### 05\_Analyze\_Dependencies / 05\_Get\_Measure\_Impact

Visualize measure dependencies. See what depends on a measure before changing it.

**You:** "What depends on Total Sales?"

**Claude:** Analyzes dependents

**Result:** 8 measures: YTD Sales, MTD Sales, Sales Growth, Profit Margin...

### 05\_Export\_DAX\_Measures / 05\_Column\_Usage\_Mapping

Export all measures to CSV. Analyze which measures use which columns.

## 06 Analysis

Model analysis and best practices

## 06\_Simple\_Analysis

Fast model overview in 2-5 seconds.

**You:** "Give me an overview"

**Claude:** Runs simple analysis

**Result:** 15 tables, 42 measures, 18 relationships. Largest: Sales (2.1M rows)

## 06\_Full\_Analysis

Comprehensive analysis with 120+ best practice rules.

**You:** "Check my model for issues"

**Claude:** Runs 120+ rules

**Result:** 3 warnings: (1) Missing description, (2) Bidirectional relationship...

## 06\_Compare\_PBI\_Models

Compare two models to see differences.

## 07 PBIP Analysis

Offline analysis without Power BI running!

### TIP

**What is PBIP?** Power BI Project format stores your model as text files. These tools analyze PBIP files **without needing Power BI Desktop running!**

### INFO

**Requirement:** PBIP tools require a **PBIP Path folder** to be specified. This is the folder containing your .pbip file and the associated Report/SemanticModel subfolders.

## 07\_Analyze\_PBIP\_Repository / 07\_Report\_Info / 07\_PBIP\_Dependency\_Analysis

Generate HTML reports, get page/visual info, explore dependencies interactively.

## 07\_Slicer\_Operations

Manage slicer configurations and visual interactions.

<code>list</code>	Find slicers with values
<code>configure_single_select</code>	Change to single-select
<code>list_interactions</code>	Cross-filtering settings
<code>set_interaction</code>	Configure interactions

## 07\_Visual\_Operations

Comprehensive visual editing for PBIP reports.

<code>list</code>	Find visuals by title/type/page
<code>update_position</code>	Move and resize visuals
<code>replace_measure</code>	Bulk replace measures
<code>sync_visual</code>	Sync across pages
<code>update_visual_config</code>	Update fonts, colors, axis

**You:** "Resize all cards to 200x100"

**Claude:** Updates all cards

**Result:** Updated 12 card visuals across 5 pages



**You:** "Replace 'Old Measure' with 'New Measure'"

**Claude:** Bulk replaces

**Result:** Replaced in 8 visuals on 4 pages

## 07\_Analyze\_Aggregation / 07\_Analyze\_Bookmarks / 07\_Analyze\_Theme\_Compliance

Analyze aggregation tables, bookmarks, and theme compliance.

### 08 Documentation

Auto-generate comprehensive model documentation

## 08\_Generate\_Documentation\_Word

Create comprehensive Word document with full model documentation including tables, measures, relationships, and data lineage.

**You:** "Generate documentation for my model"

**Claude:** Analyzes entire model, generates structured Word document

**Result:** Generated 45-page document with TOC, table descriptions, all 42 measures with DAX, relationship diagram, and data source inventory

## 08\_Generate\_Markdown\_Docs

Generate markdown documentation suitable for wikis, GitHub, or Confluence.

**You:** "Create markdown docs for our wiki"

**Claude:** Generates markdown files with proper formatting

**Result:** Created: README.md, TABLES.md, MEASURES.md, RELATIONSHIPS.md - ready for Git!

## 09 Debug Tools

Powerful debugging and analysis tools

### INFO

The Debug tools are your secret weapon for troubleshooting complex Power BI issues. From visual filter contexts to measure performance comparisons, these tools help you understand **exactly** what's happening under the hood.

### INFO

**Requirements:** Debug tools require: (1) An **open Power BI Desktop file (.pbix)** for live model connection, and (2) A **link to the PBIP report folder** to access visual and page definitions.

## 09\_Debug\_Visual - Visual Debugging

Ever wondered why a visual shows unexpected values? Debug\_Visual reveals everything: the filter context, applied slicers, cross-filters, and the actual DAX query being executed.

### Visual Debugging Flow:



### Debug\_Visual Operations

Comprehensive visual debugging with multiple analysis modes.

<code>get_filter_context</code>	See ALL filters affecting a visual (page, report, slicer, cross-filter)
<code>get_visual_query</code>	Capture the exact DAX query Power BI generates for the visual
<code>analyze_performance</code>	Measure query execution time and identify bottlenecks
<code>explain_calculation</code>	Step-by-step explanation of how a value is calculated

**You:** "Why does my Sales chart show \$0 for Electronics?"

**Claude:** Debugs the visual, reveals a hidden page-level filter excluding Electronics category

**Result:** Found the issue! Page filter 'Category NOT IN (Electronics)' is active. This filter was set on the page and is hiding Electronics from all visuals.

**You:** "Debug the filter context on my YTD Sales card"

**Claude:** Captures complete filter context including all slicer states

**Result:** Active filters: Year=2024, Region=North America, Product Category=ALL (no filter).  
Cross-filter from Customer table: Customer Segment=Enterprise. Page filter: Is Active=TRUE

## 09\_Compare\_Measures - Side-by-Side Analysis

The ultimate tool for validating optimizations. Compare two measure versions with identical filter contexts to verify they produce the same results - and see performance gains.

### Measure Comparison Flow:



Scenario	What You Learn
Before/after optimization	Verify same result, see speed improvement (e.g., 145ms vs 45ms)
Two different approaches	Which formula is faster for your data size?
Legacy vs new measure	Safe to replace? Results match exactly?
CALCULATE vs SUMX	Context transition impact on your specific model

**You:** "Compare my original Profit Margin with the optimized version"

**Claude:** Runs both measures with identical filter context, measures execution time

**Result:** Results MATCH: Both return 0.423 (42.3%) Performance: Original=145ms, Optimized=45ms - 3.2x faster! Safe to deploy the optimized version.

## 09\_Validate - Automated Testing

Run automated validation tests on your model. Catch issues before they reach production: circular dependencies, invalid references, broken relationships, and more.

### Validation Categories

Comprehensive model validation across multiple dimensions.

<code>syntax</code>	Validate all DAX expressions compile correctly
<code>references</code>	Check for broken column/table references
<code>relationships</code>	Detect circular paths, ambiguous relationships
<code>security</code>	Verify RLS/OLS rules are properly configured
<code>performance</code>	Identify potential performance issues

**You:** "Validate my model before deployment"

**Claude:** Runs comprehensive validation suite

**Result:** Validation complete: 2 warnings, 1 error - ERROR: Measure 'Old Revenue' references deleted column 'Sales[OldPrice]' - WARNING: Bidirectional relationship Date<->Sales may cause ambiguity - WARNING: 3 measures have no description

## 09\_Profile - Performance Profiling

Deep performance analysis at the query level. See exactly where time is spent: Storage Engine (SE) vs Formula Engine (FE), cache hits, and query plans.

### Performance Profiling Flow:



Metric	What It Tells You
Total Duration	End-to-end query execution time
SE (Storage Engine)	Time scanning/aggregating data from VertiPaq
FE (Formula Engine)	Time calculating DAX logic (often the bottleneck!)
SE Queries	Number of data requests (fewer is better)
Cache Hit	Was result served from cache? (warm vs cold)

**You:** "Profile my slowest measure"

**Claude:** Executes with profiling enabled, breaks down timing

**Result:** Measure: Complex YTD Calculation Total: 2,340ms | SE: 180ms (8%) | FE: 2,160ms (92%) Bottleneck: Formula Engine - the DAX is doing heavy row-by-row iteration.  
Recommendation: Replace SUMX with CALCULATE + FILTER for better SE utilization.

## 09\_Advanced\_Analysis - Deep Dive

Advanced analytical tools for complex debugging scenarios. Trace calculation chains, analyze memory usage, and understand VertiPaq compression.

### Advanced Analysis Operations

Deep technical analysis for expert users.

<code>memory_analysis</code>	Table/column memory footprint and compression ratios
<code>calculation_chain</code>	Trace the full dependency chain of a measure
<code>cardinality_analysis</code>	Find high-cardinality columns impacting performance
<code>relationship_paths</code>	Visualize all paths between two tables

**You:** "Why is my model so large? Analyze memory usage"

**Claude:** Scans all tables and columns, calculates compression ratios

**Result:** Total model size: 2.4GB Top consumers: 1. FactTransactions[Description] - 890MB (37%) - High cardinality text! 2. FactTransactions[TransactionID] - 450MB (19%) - GUID column 3. DimProduct[ProductDetails] - 280MB (12%) Recommendation: Remove Description column or move to separate table.

**You:** "Show me all unused columns in my model"

**Claude:** Analyzes report visuals and measure references

**Result:** Found 23 unused columns across 8 tables: - DimCustomer: [OldAddress], [Fax], [MiddleName] - FactSales: [LegacyID], [ImportDate], [BatchNumber] - DimProduct: [DiscontinuedDate], [OldSKU]... Removing these could reduce model size by ~180MB.

## 10 Help & Reference

Built-in guidance and documentation

### 10\_Show\_User\_Guide

Access built-in documentation, tool reference, and examples directly from Claude.

**You:** "How do I use the DAX Intelligence tool?"

**Claude:** Shows detailed usage guide with examples

**Result:** DAX Intelligence accepts measure names or DAX expressions. Use 'analyze' mode for context analysis, 'debug' for step-by-step...

# 7. Common Workflows

---

## Workflow 1: Model Exploration



### EXAMPLE

1. "Connect to my Power BI model"
2. "Give me an overview"
3. "List all tables"
4. "Describe the Sales table"
5. "Show me all measures"

## Workflow 2: Creating Measures



### EXAMPLE

1. "Show existing sales measures"
2. "Create a measure for YoY growth"
3. "Analyze this measure for issues"
4. "Test it: show growth by category"
5. Refine if needed

## Workflow 3: Performance



### EXAMPLE

1. "Run full performance analysis"
2. "Show high cardinality columns"
3. "Analyze the slowest measure"
4. "Compare original vs optimized"
5. "Update the measure"

# 8. Tips & Best Practices

## NOTE

- **Start simple** - "Connect" then "Give me an overview"
- **Use natural language** - Claude picks the right tool
- **Be specific** - "Show DAX for Total Sales" not "show code"
- **Iterate** - Ask follow-up questions
- **Analyze before changing** - Check impact first

## IMPORTANT

- **Don't close Power BI** while using MCP
- **Don't skip testing** - Verify changes work
- **Don't ignore warnings** - Best practice results matter
- **Always connect first** before other operations

## Quick Reference

You Say...	What Happens
"What's in my model?"	Quick overview of tables, measures, relationships
"Show DAX for [name]"	Retrieves full measure expression
"Create a measure for..."	Claude writes DAX and creates it
"What's wrong with this?"	DAX analysis with suggestions
"Make my model faster"	Performance analysis with recommendations
"Document my model"	Generates Word documentation
"What depends on [X]?"	Impact analysis before changes

## 9. Beyond Power BI: Claude as Your Dev Partner

### INFO

Claude isn't just for Power BI analysis! When working with Power BI projects in a development environment, Claude can assist with **Visual Studio workflows**, **deployment conflicts**, **Microsoft Fabric notebooks**, and **PySpark** development.

### Visual Studio & Source Control

When working with PBIP (Power BI Project) files in Visual Studio or VS Code, Claude can help with:

- **Understanding TMDL files** - Claude can explain table definitions, measures, and relationships
- **Code reviews** - Review changes before committing to version control
- **Debugging issues** - Identify problems in model definition files
- **Refactoring** - Suggest improvements to measure organization and naming

**You:** "I'm looking at model.tmdl in Visual Studio - explain what this partition definition does"

**Claude:** Reads the TMDL structure and explains the partition type, source query, and refresh behavior

**Result:** This is an M-partition querying your SQL database with incremental refresh enabled...

### Resolving Merge Conflicts in Deployments

**Merge Conflict Resolution Flow:**



When multiple developers work on the same Power BI project, merge conflicts can occur. Claude helps you understand and resolve these conflicts:

- **Identify conflicting changes** - Understand what each version modified
- **Semantic analysis** - Know if changes affect the same measures/relationships
- **Resolution strategies** - Get guidance on which changes to keep
- **Impact assessment** - Understand downstream effects of each choice



**You:** "I have a merge conflict in Sales.tmdl between dev and main branch - help me resolve it"

**Claude:** Analyzes both versions, identifies the conflicting measure definitions, explains the differences

**Result:** Branch 'dev' added a new filter to [Total Sales], while 'main' updated the format string. These changes are compatible - you can keep both modifications.

### TIP

**Pro Tip:** Before deploying, ask Claude to compare your development model with production using 06\_Compare\_PBI\_Models to preview all differences and potential conflicts.

## Microsoft Fabric Notebooks

Microsoft Fabric notebooks allow you to work with data using Python, PySpark, and SQL. Claude can assist with writing and debugging notebook code:

- **Data transformation** - Write efficient Spark transformations
- **Lakehouse integration** - Query Delta tables and manage data pipelines
- **Semantic model refresh** - Script automated refreshes via the Fabric REST API
- **Cross-workspace operations** - Work with data across multiple workspaces

**You:** "Write a Fabric notebook cell that reads from my Lakehouse and aggregates sales by region"

**Claude:** Writes PySpark code using `spark.read.format('delta')` with proper aggregations

**Result:** `df = spark.read.format('delta').load('Tables/Sales')\nresult = df.groupBy('Region').agg(sum('Amount').alias('TotalSales'))`

## PySpark Development

### PySpark Development Flow:



PySpark is essential for big data processing in Fabric. Claude helps with:

- **Writing efficient transformations** - Avoid common performance pitfalls
- **Schema management** - Define and evolve Delta table schemas
- **Window functions** - Complex analytical queries made simple
- **UDF optimization** - When to use and when to avoid user-defined functions
- **Debugging errors** - Understand cryptic Spark error messages

You Ask...

Claude Helps With...

"Calculate running total by date"	Window function with orderBy and rowsBetween
"Deduplicate records keeping latest"	Row numbering with partition and filter
"Join two large DataFrames efficiently"	Broadcast hints, bucketing strategies
"Read from SQL Server incrementally"	JDBC connection with partition predicates
"Write to Delta with merge/upsert"	DeltaTable.forPath with merge conditions

## NOTE

**The Power Combo:** Use the MCP-PowerBI-Finvision server to analyze your semantic model, then ask Claude to write PySpark code in Fabric notebooks that prepares data specifically for that model. Full end-to-end assistance!

# 10. The Future: Fabric MCP Server

## INFO

Imagine the power of MCP-PowerBI-Finvision... but for the **entire Microsoft Fabric ecosystem**. A dedicated **Fabric MCP Server** could revolutionize how you interact with your data platform!

## The Vision

A Fabric MCP Server would enable Claude to directly interact with all Fabric workloads - from Lakehouses to Data Pipelines, from Notebooks to Real-Time Analytics. Natural language becomes your interface to the entire data platform.

### End-to-End Fabric Automation:



## Potential Capabilities

### Lakehouse Operations

- **Table management** - Create, alter, optimize Delta tables via natural language
- **Data exploration** - "Show me the schema of the Sales table"
- **Quality checks** - "Are there any null values in the CustomerID column?"
- **Time travel** - "Show me the data as it was yesterday at 3pm"

### Data Pipeline Orchestration

- **Pipeline monitoring** - "What pipelines failed in the last 24 hours?"
- **Trigger management** - "Pause all pipelines in the Finance workspace"
- **Lineage tracking** - "Where does this table get its data from?"
- **Cost analysis** - "Which pipelines consume the most capacity?"

### Notebook Automation

- **Code generation** - "Write a notebook that deduplicates the Orders table"
- **Execution** - "Run the daily-refresh notebook and tell me when it finishes"
- **Debugging** - "Why did this notebook fail? Show me the error"
- **Scheduling** - "Schedule this notebook to run every Monday at 6am"

### Warehouse & SQL Analytics

- **Query execution** - "Show me top 10 customers by revenue this quarter"
- **Schema exploration** - "What views depend on the DimCustomer table?"
- **Performance tuning** - "Which queries are running slow today?"
- **Security audit** - "Who has access to the PII schema?"

# What Conversations Could Look Like

**You:** "Create a new Lakehouse table from the CSV files in the landing zone"

**Claude:** Scans landing zone, infers schema, creates optimized Delta table with proper partitioning

**Result:** Created 'Sales\_Raw' table with 2.3M rows, partitioned by Year/Month, Z-ordered by CustomerID

**You:** "Why is the Finance pipeline taking so long today?"

**Claude:** Analyzes run history, checks data volumes, identifies bottleneck activity

**Result:** The 'Transform\_Transactions' activity is processing 3x more rows than usual due to month-end. Consider adding incremental refresh.

**You:** "Set up a complete ETL from our Azure SQL database to a Gold layer table"

**Claude:** Creates connection, designs pipeline with bronze/silver/gold architecture, implements SCD Type 2

**Result:** Created pipeline 'SQL\_to\_Gold\_Customers' with: source connection, bronze landing, silver cleansing, gold dimension table with history tracking. Ready for scheduling.

## The Combined Power: Finvision + Fabric

Current: Finvision	Future: + Fabric MCP
Analyze semantic model	Analyze + trace data back to source
Optimize DAX measures	Optimize DAX + underlying Lakehouse queries
Document Power BI model	Document entire data lineage end-to-end
Debug visual filter context	Debug from visual through pipeline to source
Create measures in Power BI	Create full stack: source query + transform + measure

**TIP**

**Stay Tuned!** The Fabric MCP Server will be released as a standalone companion project. It will transform how teams interact with their entire Microsoft Fabric environment - making AI assistance available across the full data lifecycle, from ingestion to insights.

## **MCP-PowerBI-Finvision** | AI-Powered Power BI Analysis

For more info, visit the [project repository](#) or ask Claude!