P2P Exchange Rate Analysis

From Data Pipeline to Dashboard

Ignacio Velez Ocampo Justiniano   Olivier Manthey
Alessandro Pizzi   Axel Lo Schiavo

2025-12-01

Introduction

This document provides comprehensive documentation for our final project in the Data and Code Management course. The project addresses a real-world data challenge: tracking and analyzing parallel currency markets in countries experiencing foreign exchange constraints, with a focus on Bolivia.

The project is organized into three phases, each building upon the previous:

Phase 1 — Data Gathering and Wrangling: An automated pipeline that collects exchange rate data from Binance P2P (market-implied rates) and the Central Bank of Bolivia (official rates) every 15 minutes, storing raw and processed data in Parquet format.

Phase 2 — Python Analytics Package: A modular Python package (`p2p-analytics`) that transforms raw data into analysis-ready outputs through single-line function calls, enabling computation of spreads, premiums, volatility, and market microstructure metrics.

Phase 3 — Interactive Dashboard: A Streamlit web application that visualizes the analytical outputs, deployed to a public URL for user-facing exploration without requiring code execution.

Together, these phases demonstrate a complete data science workflow: from raw data acquisition through analytical processing to interactive visualization.

Background and Motivation

The Bolivian Currency Crisis

Bolivia has maintained a fixed exchange rate regime since 2011, with the official rate pegged at approximately 6.96 Bolivianos (BOB) per US dollar. This policy provided stability for over a decade, but beginning in 2023, the country experienced a severe foreign currency crisis that exposed the limitations of administratively set exchange rates.

The crisis originated from a combination of factors. Bolivia's foreign currency reserves declined precipitously, falling from approximately 15 billion USD in 2014 to under 2 billion USD by 2024. This decline resulted from reduced natural gas exports (the country's primary source of foreign exchange), increased import dependence, and sustained capital outflows. As reserves dwindled, the Central Bank found itself unable to supply dollars at the official rate to meet demand.

The practical consequence is that while the official exchange rate remains fixed at 6.96 BOB/USD, Bolivians seeking to purchase dollars must often turn to parallel markets where prices have ranged from 9 to 20 BOB/USD depending on market conditions. This gap between official and parallel rates represents a "currency premium" that reflects the true scarcity of foreign exchange.

The Role of Peer-to-Peer Cryptocurrency Markets

In this environment, cryptocurrency peer-to-peer (P2P) platforms have emerged as a significant channel for obtaining dollar-equivalent assets. Binance P2P allows users to trade stablecoins such as USDT (Tether, pegged 1:1 to the US dollar) directly with other users using local bank transfers.

The mechanism operates as follows: a seller posts an advertisement offering to sell USDT at a specified price in local currency. A buyer accepts the offer and transfers fiat currency through local banking channels. Binance holds the seller's cryptocurrency in escrow during the transaction, releasing it only after the seller confirms receipt of payment.

For our purposes, the P2P market serves as a proxy for the "true" market exchange rate. When a Bolivian user purchases USDT at 9.50 BOB per unit, this effectively represents an exchange rate of 9.50 BOB/USD, regardless of what the official rate states.

Why Build This Project?

Binance does not provide historical P2P price data through its platform. Users can view current advertisements, but cannot query what prices were available yesterday, last week, or last month. This limitation creates a data gap that our project fills.

By collecting snapshots of the P2P market at regular intervals, we build a historical database that enables:

- Parallel premium analysis comparing P2P rates to official rates
- Spread analysis examining bid-ask gaps as liquidity indicators
- Volatility measurement tracking price stability
- Cross-currency comparison across different fiat markets

Inclusion of other currencies

While Bolivia (BOB) is the primary focus of this project, we deliberately included seven additional currencies to enhance the robustness and generalizability of our pipeline. Argentina (ARS) serves as a natural comparison case, another Latin American economy experiencing currency stress and active parallel markets. The major international currencies (USD, EUR, GBP) act as stable benchmarks, allowing us to verify that our analytics behave correctly under low-volatility conditions. Mexico (MXN) provides a regional reference point with a liquid, freely-traded currency. Finally, Japan (JPY) and China (CNY) extend coverage to Asian markets, with CNY being particularly relevant due to its capital controls, a structural similarity to Bolivia's situation. This multi-currency design ensures the package is not overfitted to a single market and can be readily extended to other countries facing similar exchange rate pressures.

Phase 1 — Data Gathering and Wrangling

Phase 1 establishes the foundation of the project: an automated data pipeline that continuously collects, cleans, and stores exchange rate data from two complementary sources.

Data Sources

Source 1: Binance P2P Marketplace

Binance P2P is a peer-to-peer trading platform operated by Binance, one of the world's largest cryptocurrency exchanges. The platform facilitates direct trades between users, with Binance providing escrow services to ensure transaction security.

We access Binance P2P data through a public API endpoint that does not require authentication:

```
POST https://p2p.binance.com/bapi/c2c/v2/friendly/c2c/adv/search
```

The request payload specifies the cryptocurrency (`USDT`), fiat currency, trade direction (`BUY` or `SELL`), and pagination parameters. The API returns advertisement details including price, amounts, merchant information, and payment methods.

Data Fields Collected

Field	Type	Description
`timestamp_scraped`	datetime	UTC timestamp when data was collected
`side`	string	Trade direction: "BUY" or "SELL"
`price`	float	Price per USDT in fiat currency
`asset`	string	Cryptocurrency (always "USDT")
`fiat`	string	Fiat currency code
`min_amount`	float	Minimum transaction amount in fiat
`max_amount`	float	Maximum transaction amount in fiat
`merchant_id`	string	Unique identifier for the trader
`merchant_name`	string	Display name of the trader
`finish_rate`	float	Proportion of trades completed (0-1)
`positive_rate`	float	Proportion of positive feedback (0-1)
`payment_methods`	string	Accepted payment methods

Currencies Tracked

Currency	Country/Region	Rationale
BOB	Bolivia	Primary focus; subject to currency crisis
ARS	Argentina	High-inflation economy with parallel markets
USD	United States	Global reference currency
EUR	Eurozone	Major international currency
GBP	United Kingdom	Major international currency
MXN	Mexico	Large Latin American economy
JPY	Japan	Asian reference currency
CNY	China	Large economy with capital controls

Each pipeline run collects the top 40 BUY and 40 SELL advertisements for each currency, yielding approximately 640 observations before deduplication.

Source 2: Central Bank of Bolivia (BCB)

The Banco Central de Bolivia publishes official exchange rates on its website. These rates represent the administratively set prices at which the government values foreign currencies.

We scrape the BCB exchange rate table using HTTP requests and HTML parsing:

```
URL: https://www.bcb.gob.bo/librerias/indicadores/otras/ultimo.php
Method: HTML table extraction via pandas.read_html()
```

Data Fields Collected

Field	Type	Description
`date`	date	Publication date
`currency`	string	Currency code
`official_exchange_rate`	float	Official rate published by BCB

The BCB publishes rates once per day. Our pipeline checks whether the publication date has changed before scraping, avoiding duplicate entries.

Terms of Use and Ethical Considerations

Binance API Usage

- We access only public, unauthenticated endpoints
- We implement rate limiting (0.5-second delays between requests)
- Data is used exclusively for academic purposes
- The pipeline runs every 15 minutes, representing modest load

BCB Data Usage

The Central Bank of Bolivia publishes exchange rates as public government information intended for transparency. Our academic usage aligns with this purpose.

Data Privacy

Merchant names and identifiers are voluntarily published by traders on a public platform. We do not collect transaction histories or non-public user data.

Pipeline Architecture

The pipeline follows a standard ETL pattern with automated scheduling:

```
+---------------------------------------------------------------------+
|                        CRON-JOBS.ORG                                |
|                    External Scheduler                               |
|                  Triggers every 15 minutes                          |
+---------------------------------------------------------------------+
                                 |
                                 v
+---------------------------------------------------------------------+
|                     GITHUB ACTIONS                                  |
|                 phase1_data_pipeline.yml                            |
|                                                                     |
|   1. Checkout repository                                            |
|   2. Setup Python 3.12                                              |
|   3. Install dependencies                                           |
|   4. Execute run_pipeline.py                                        |
|   5. Commit and push updated data files                             |
+---------------------------------------------------------------------+
                                 |
                                 v
+---------------------------------------------------------------------+
|                    run_pipeline.py                                  |
|                                                                     |
|   - Verify internet connectivity                                    |
|   - Execute Binance pipeline (update_binance)                       |
|   - Execute BCB pipeline (update_bcb)                               |
|   - Write execution logs                                            |
+---------------------------------------------------------------------+
                                 |
                    +------------+------------+
                    |                         |
                    v                         v
     +--------------------------+  +--------------------------+
     |    BINANCE PIPELINE      |  |      BCB PIPELINE        |
     |                          |  |                          |
     |  Runs every 15 minutes   |  |  Runs daily (if new)     |
     +--------------------------+  +--------------------------+
```

Repository Structure:

```
phase1_data_pipeline/
|
+-- scripts/
|   +-- run_pipeline.py              # Main entry point
|   +-- binance/
|   |   +-- paths_binance.py         # Path configuration
|   |   +-- base_scraper.py          # Core API functions
|   |   +-- multi_fetch.py           # Multi-page fetching
|   |   +-- save_raw.py              # Raw data storage
|   |   +-- clean_standardize.py     # Data cleaning
|   |   +-- snapshot_and_master.py   # Output management
|   |   +-- update_binance.py        # Orchestration
|   +-- bcb/
|       +-- scrape_bcb.py            # HTML scraping
|       +-- clean_bcb.py             # Numeric cleaning
|       +-- extract_bcb.py           # Rate extraction
|       +-- update_bcb.py            # Orchestration
|
+-- data/
|   +-- raw/
|   |   +-- binance/                 # Raw snapshots
|   |   +-- bcb/                     # Raw tables
|   +-- processed/
|       +-- binance/
|       |   +-- p2p_master.parquet   # Complete history
|       |   +-- daily_snapshots/     # Per-day files
|       |   +-- historical_fiat/     # Per-currency files
|       +-- bcb/
|           +-- bcb_master.parquet   # Complete history
|
+-- logs/
    +-- binance/                     # Execution logs
    +-- bcb/
```

Binance Pipeline Implementation

Path Configuration

Paths are computed relative to the script location, ensuring portability:

```python
import os

BINANCE_DIR = os.path.dirname(os.path.abspath(__file__))
SCRIPTS_DIR = os.path.dirname(BINANCE_DIR)
PHASE1_DIR = os.path.dirname(SCRIPTS_DIR)
DATA_DIR = os.path.join(PHASE1_DIR, "data")

DATA_RAW_BINANCE = os.path.join(DATA_DIR, "raw", "binance")
DATA_PROCESSED_BINANCE = os.path.join(DATA_DIR, "processed", "binance")
MASTER_PATH = os.path.join(DATA_PROCESSED_BINANCE, "p2p_master.parquet")
```

API Request with Retry Logic

```python
def safe_request(url, payload, max_retries=3, delay=2):
    """Sends a POST request with retry logic."""
    for attempt in range(max_retries):
        try:
            response = requests.post(url, json=payload, timeout=10)
            response.raise_for_status()
            return response.json()
        except Exception:
            if attempt < max_retries - 1:
                time.sleep(delay)
            else:
                return None
```

Multi-Currency Fetching

```python
def p2p_fetch(asset, fiat, run_index, pages=2, delay=0.5):
    """Fetches P2P data for a single currency."""
    sides = ["BUY", "SELL"]
    all_rows = []

    for side in sides:
        for page in range(1, pages + 1):
            json_response = p2p_query(asset, fiat, side, page=page)
            df = p2p_to_df(json_response, side)
            time.sleep(delay)  # Rate limiting

            if not df.empty:
                df["run_index"] = run_index
                all_rows.append(df)

    return pd.concat(all_rows, ignore_index=True).drop_duplicates()
```

Cleaning and Standardization

The cleaning pipeline applies several transformations:

Column normalization:

```
df.columns = [col.strip().lower() for col in df.columns]
```

Numeric conversion:

```
for col in ["min_amount", "max_amount"]:
    df[col] = (
        df[col].astype(str)
        .str.replace(",", "", regex=False)
        .str.strip()
    )
    df[col] = pd.to_numeric(df[col], errors="coerce")
```

Categorical standardization:

```
for col in ["side", "asset", "fiat"]:
    df[col] = df[col].astype(str).str.upper().str.strip()
```

Timestamp formatting:

```
df["timestamp_scraped"] = (
    pd.to_datetime(df["timestamp_scraped"], errors="coerce")
      .dt.strftime("%Y-%m-%dT%H:%M:%SZ")
)
```

Output Management

The pipeline produces three types of processed outputs:

1. **Master file**: Complete historical dataset (`p2p_master.parquet`)
2. **Daily snapshots**: One file per calendar day

3. **Per-currency files**: Separate history per fiat currency

Temporal columns are added before saving:

```python
def add_time_columns(df):
    ts = pd.to_datetime(df["timestamp_scraped"], errors="coerce")
    df["scrape_datetime"] = ts.dt.strftime("%Y-%m-%d %H:%M")
    df["date"] = ts.dt.strftime("%Y-%m-%d")
    df["time"] = ts.dt.strftime("%H:%M")
    df["year"] = ts.dt.year
    df["month"] = ts.dt.month
    df["day"] = ts.dt.day
    df["year_month"] = ts.dt.strftime("%Y-%m")
    return df
```

BCB Pipeline Implementation

Date Extraction

```python
def get_bcb_date():
    """Extracts publication date from BCB page (Spanish format)."""
    url = "https://www.bcb.gob.bo/librerias/indicadores/otras/ultimo.php"
    html = requests.get(url).text

    match = re.search(r"(\d{1,2}) de ([A-Za-z]+) (\d{4})", html)
    if not match:
        return None

    day, month_text, year = match.groups()
    months = {
        "enero": "01", "febrero": "02", "marzo": "03", "abril": "04",
        "mayo": "05", "junio": "06", "julio": "07", "agosto": "08",
        "septiembre": "09", "octubre": "10", "noviembre": "11",
        "diciembre": "12"
    }
    month_num = months[month_text.lower()]
    return f"{year}-{month_num}-{day.zfill(2)}"
```

Numeric Cleaning

The BCB uses European-style number formatting (commas as decimal separators):

```
def clean_bcb_table(df):
    for col in ["tipo_cambio_bs", "tipo_cambio_me"]:
        df[col] = (
            df[col].astype(str)
            .str.replace(",", ".", regex=False)
            .str.replace(" ", "", regex=False)
        )
        df[col] = pd.to_numeric(df[col], errors="coerce")
    return df
```

Idempotent Updates

The BCB pipeline skips execution if today's data is already stored:

```
def update_bcb():
    extracted_date = get_bcb_date()
    metadata_date = _load_metadata_date()

    if extracted_date is not None and metadata_date == extracted_date:
        print("[BCB] skip (already updated)")
        return None
    # Proceed with scraping...
```

Automation and Scheduling

GitHub Actions Workflow

```
name: Phase 1 Data Pipeline

on:
  workflow_dispatch:

jobs:
  run-pipeline:
    runs-on: ubuntu-latest
    permissions:
      contents: write

    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
```

```
        with:
          python-version: "3.12"
      - run: pip install -r requirements.txt
      - run: python phase1_data_pipeline/scripts/run_pipeline.py
      - run: |
          git config --global user.name "github-actions"
          git config --global user.email "actions@github.com"
          git add phase1_data_pipeline/data/* phase1_data_pipeline/logs/*
          git commit -m "Automated data update" || echo "No changes"
          git push
```

Resource Usage

GitHub Actions provides 2,000 free minutes per month. Our usage:

Parameter	Value
Runs per month	~2,880
Duration per run	~39 seconds
Total minutes/month	**~1,872**

This is very adjusted in relation with the 2,000-minute limit. That is one of the reasons why we could not add more currencies.

Data Schemas

Processed Binance Data

Column	Type	Description
`run_index`	int64	Sequential pipeline run identifier
`scrape_datetime`	string	"YYYY-MM-DD HH:MM"
`date`	string	"YYYY-MM-DD"
`time`	string	"HH:MM"
`year`, `month`, `day`	int	Date components
`year_month`	string	"YYYY-MM"
`currency`	string	Fiat currency code
`side`	string	"BUY" or "SELL"
`price`	float64	Price per USDT
`min_amount`, `max_amount`	float64	Transaction limits
`merchant_name`	string	Trader display name

Column	Type	Description
`finish_rate`, `positive_rate`	float64	Trader reputation

BCB Data

Column	Type	Description
`date`	string	Publication date
`currency`	string	Currency code
`official_exchange_rate`	float64	Official rate

Limitations

Top advertisements only: We collect only the top 40 BUY and 40 SELL advertisements per currency. Deeper liquidity is not captured.

15-minute frequency: Prices can change within seconds, but we sample only every 15 minutes. This is sufficient for daily trends but cannot capture high-frequency dynamics.

No volume data: The API does not provide actual trade volume. We cannot weight prices by transaction size.

Platform selection: Our data captures only Binance P2P, not other channels such as LocalBitcoins or cash markets.

Survivorship bias: We observe only currently active advertisements. Quickly matched ads are underrepresented.

BCB scraping fragility: Website structure changes could break the scraper.

Phase 2 — Python Analytics Package

Phase 2 delivers a Python package that transforms Phase 1 data into analysis-ready outputs through simple, single-line function calls.

Purpose and Design Philosophy

Phase 1 produces structured Parquet files, but working with them directly requires repetitive code for loading, filtering, and aggregating. Phase 2 addresses this friction by encapsulating common operations into a clean, modular package.

The guiding principle is **single-line usability**: each analytical task should require only one function call.

```
from pathlib import Path
PHASE1_ROOT = Path("../final-project/phase1_data_pipeline/data/processed").resolve()

from p2p_analytics import p2p_spread
spread_ars = p2p_spread("ARS", by="day", root=PHASE1_ROOT)
```

All functions return pandas DataFrames, enabling immediate export:

```
p2p_summary("ARS", root=PHASE1_ROOT).to_csv("exports/summary_ARS.csv")
```

Package Structure

```
phase2_package/
|
+-- src/p2p_analytics/
|   +-- __init__.py          # Public API exports
|   +-- io.py                # Data loading
|   +-- spreads.py           # Spread analysis
|   +-- premium.py           # Official rate comparison
|   +-- microstructure.py    # Order flow analysis
|   +-- summary.py           # Aggregated statistics
|
+-- tests/
|   +-- conftest.py          # Shared fixtures
|   +-- test_io.py
|   +-- test_spreads.py
|   +-- test_summary.py
|
+-- docs/                    # MkDocs documentation
+-- mkdocs.yml
+-- pyproject.toml
```

Module Overview

io Module — Data Loading

Function	Purpose
`get_processed_root()`	Infer path to processed data directory
`load_binance_master()`	Load complete Binance history
`load_binance_currency(currency)`	Load single currency history
`load_binance_daily(date)`	Load single day snapshot
`load_bcb_master()`	Load complete BCB history
`load_bcb_latest()`	Load latest BCB rates

```python
from pathlib import Path
PHASE1_ROOT = Path("../final-project/phase1_data_pipeline/data/processed").resolve()

from p2p_analytics import load_binance_currency, load_bcb_master

df_ars = load_binance_currency("ARS", root=PHASE1_ROOT)
bcb = load_bcb_master(root=PHASE1_ROOT)
```

spreads Module — Spread Analysis

Function	Purpose
`p2p_spread(currency, by)`	Daily or hourly bid-ask spread
`intraday_profile(currency)`	Average prices by hour of day
`fiat_comparison(currencies)`	Multi-currency daily comparison

```python
from p2p_analytics import p2p_spread, intraday_profile

spread = p2p_spread("ARS", by="day", root=PHASE1_ROOT)
profile = intraday_profile("BOB", root=PHASE1_ROOT)
```

Output columns for `p2p_spread`: `date`, `currency`, `avg_buy_price`, `avg_sell_price`, `spread_abs`, `spread_pct`

premium Module — Official Rate Comparison

Function	Purpose
`official_premium(currency, by)`	Premium vs BCB official rate

```python
from p2p_analytics import official_premium

premium = official_premium("BOB", root=PHASE1_ROOT)
# Shows how much more expensive P2P is vs official rate
```

Output columns: `date`, `currency`, `p2p_avg_price`, `official_exchange_rate`, `premium_abs`, `premium_pct`

microstructure Module — Order Flow

Function	Purpose
`order_imbalance(currency, by)`	Buy/sell volume imbalance

```python
from p2p_analytics import order_imbalance

imbalance = order_imbalance("BOB", by="hour", root=PHASE1_ROOT)
```

Imbalance formula: $\frac{\text{buy\_volume}-\text{sell\_volume}}{\text{buy\_volume}+\text{sell\_volume}}$

summary Module — Statistics and Rankings

Function	Purpose
`p2p_summary(currency, by)`	Comprehensive daily/hourly statistics
`top_advertisers(currency, n)`	Top merchants by volume
`price_volatility(currency, window)`	Rolling volatility of mid-price

```python
from p2p_analytics import p2p_summary, top_advertisers, price_volatility

summary = p2p_summary("ARS", by="day", root=PHASE1_ROOT)
top = top_advertisers("BOB", n=10, root=PHASE1_ROOT)
vol = price_volatility("ARS", window=7, root=PHASE1_ROOT)
```

Installation

```
cd final-project
python -m venv .venv
source .venv/bin/activate  # or .\.venv\Scripts\activate on Windows
pip install -e ../p2p-analytics/phase2_package
```

Testing

Tests use pytest with synthetic fixtures:

```
pytest -v
```

Example test:

```
def test_p2p_spread_core_daily(sample_binance_df):
    result = _p2p_spread_core(sample_binance_df, "ARS", by="day")
    assert "spread_abs" in result.columns
    expected_spread = 1510 - 1500
    assert abs(result["spread_abs"].iloc[0] - expected_spread) < 0.01
```

Documentation Website

MkDocs generates a documentation site with API reference from docstrings:

```
mkdocs serve  # Local preview at http://127.0.0.1:8000
mkdocs build  # Generate static HTML
```

Limitations

Scope: The package implements a curated set of functions rather than exhaustive coverage. This was deliberate to ensure quality within time constraints.

Currency coverage: Functions assume the eight currencies from Phase 1.

Static data: The package reads from files; it does not integrate with live streams.

Separate repository structure: Building the package in a separate folder from the data pipeline requires explicit `root` path configuration in all function calls. This makes setup less seamless than having everything in a single repository, but was chosen to maintain cleaner separation of concerns between data collection (Phase 1) and analytics (Phase 2).

Phase 3 — Interactive Dashboard

Phase 3 transforms the analytical outputs into a user-facing Streamlit dashboard deployed to a public URL.

From Analysis to Product

Phase 3 represents a fundamental shift: from code that analysts run to a product that anyone can explore. The dashboard makes insights accessible without requiring Python knowledge.

A key architectural decision was to **separate computation from presentation**:

1. Phase 2 functions compute metrics
2. Results are exported to CSV files
3. The Streamlit app loads and visualizes CSVs

This separation ensures fast, responsive interactions (no recomputation on every click) and makes the CSVs portable to other tools like Power BI.

Application Architecture

Streamlit reruns the entire script on every interaction. To manage complexity, we adopted a modular structure:

```
streamlit_app/
|
+-- P2P_Binance.py           # Main entry point
+-- app/
|   +-- data.py              # Data loading with caching
|   +-- viz.py               # Visualization functions
|   +-- layout.py            # UI helpers
+-- pages/
    +-- 1_Spread_Overview.py
    +-- 2_Insights_by_Currency.py
    +-- 3_Summary_Table.py
```

data.py: Centralizes loading with `@st.cache_data` for performance.

viz.py: Creates Plot figures from DataFrames.

layout.py: Provides consistent UI patterns (headers, selectors, metric cards).

Dashboard

Page	Purpose	Visualization
Spread Overview	Compare bid-ask spreads across currencies	Line chart over time (daily) by currency
Intraday Profile	Reveal hourly price patterns in BUY vs SELL prices for a selected currency	Line chart (BUY vs SELL) by hour
Official Premium	Quantify the gap between P2P rate and official exchange rate	Line chart of premium (%, and absolute) over time
Order Imbalance	Detect demand/supply dominance (BUY vs SELL pressure)	Heatmap of imbalance
Spread Analysis	Track market liquidity via the P2P bid–ask spread	Heatmap of spread
Price Volatility	Highlight regime changes in price variability	Rolling (7-day) volatility time-series
Top Advertisers	Identify dominant market participants and concentration	Horizontal bar ranking (top N advertisers)
Summary Table	Provide an at-a-glance view of computed metrics per date/currency	Interactive data table (filter/sort)

Data Export Process

A Python script (`data_extraction.py`) generates all CSV exports and this has been included in the workflow that runs automatically the data pipeline of phase 1. The idea is to feed Streamlit with the latest information available.

```python
from p2p_analytics import p2p_spread, intraday_profile, official_premium
```

Deployment

The dashboard is deployed via Streamlit Cloud:

1. Push code to GitHub
2. Connect repository to Streamlit Cloud
3. Specify entry point: `phase3_dashboard/streamlit_app/P2P_Binance.py`
4. Deploy

Limitations

Data freshness: The dashboard displays pre-computed CSVs, not live data. Updates require re-running the export notebook.

Short time window: Limited historical data constrains trend analysis.

Streamlit constraints: Some advanced UI patterns are difficult to implement; customization has limits.

Conclusion

This project demonstrates a complete data science workflow applied to a real-world problem: understanding parallel currency markets through P2P exchange data.

Phase 1 established the foundation with an automated pipeline collecting data every 15 minutes from Binance P2P and the Central Bank of Bolivia. The pipeline is fully reproducible, storing raw and processed data in version-controlled Parquet files.

Phase 2 built an analytics layer that transforms raw data into insights through simple function calls. The modular package design ensures maintainability and testability.

Phase 3 delivered a user-facing product—an interactive dashboard accessible via public URL. The separation of computation and presentation ensures performance and portability.

The project addresses a genuine data gap (Binance's lack of historical P2P data) and produces outputs relevant to understanding currency stress in Bolivia and similar economies. As data accumulates over time, the analytical value will grow, enabling more robust trend analysis and event studies.

Appendix A: Sample Execution Log

```
Running Binance Pipeline
Start (UTC): 2025-12-08 22:00:56.100337+00:00
End (UTC): 2025-12-08 22:01:20.176746+00:00
Duration: 24.08 seconds
Status: success

Running BCB Pipeline
Start (UTC): 2025-12-08 22:01:20.200000+00:00
End (UTC): 2025-12-08 22:01:21.500000+00:00
Duration: 1.3 seconds
Status: success
```

Appendix B: Dependencies

Phase 1 (Pipeline):

```
pandas>=2.0.0
requests>=2.31.0
pyarrow>=14.0.0
lxml>=4.9.0
```

Phase 2 (Package):

```
pandas
numpy
pyarrow
matplotlib
pytest (dev)
mkdocs (dev)
```

Phase 3 (Dashboard):

```
ipykernel
streamlit
pandas
altair
matplotlib
```

Appendix C: Data Sources

- **Binance P2P**: https://p2p.binance.com
- **Banco Central de Bolivia**: https://www.bcb.gob.bo