404error

1. Lee Ming Xuan
2. Goh Sheng Kai
3. Thim Yee Song

Cash Flow Intelligence: Predicting Liquidity with Advanced ML

A 6-Month Rolling Forecast & Anomaly Detection System for AstraZeneca

Data is the new oil. Insight is the new currency

Business Context & Objective

The Challenge

AstraZeneca operates across complex regional networks where varying billing cycles and fragmented data create significant **blind spots** in short-term cash visibility. Finance teams currently lack the **automated tools** needed to quickly identify and triage irregular transactions amidst this noise.

Our Objective

Our solution aims to bridge this gap by building a machine learning engine that delivers reliable weekly forecasts for the next six months. Simultaneously, we provide an automated risk detection system to instantly flag liquidity anomalies for business review.

Data Strategy

Data Consolidation & Cleansing

- Multi-Table Join: Architected a relational join between Main Data, Country Mapping, and Category Linkage to stabilize entity-level granularity.
- USD Stabilization: Synchronized all multi-currency transactions into USD using provided exchange rates to eliminate FX translation noise.
- Critical Sanitization: Corrected structural typos (e.g., 'Non Netting AP' vs 'Non-Netting AP') and applied conditional sign logic for the Other category based on transaction directionality.

EDA & Outlier Profiling

- **Skewness Analysis:** 10% of high-value transactions drive 80% of total liquidity risk—focusing model weights on high-impact rows.
- **Anomaly vs. Cycle:** Distinguished between random noise and predictable structural outliers (Quarter-end spikes).

Data Strategy & Transformation

Phase 1: Harmonization & Quality Control

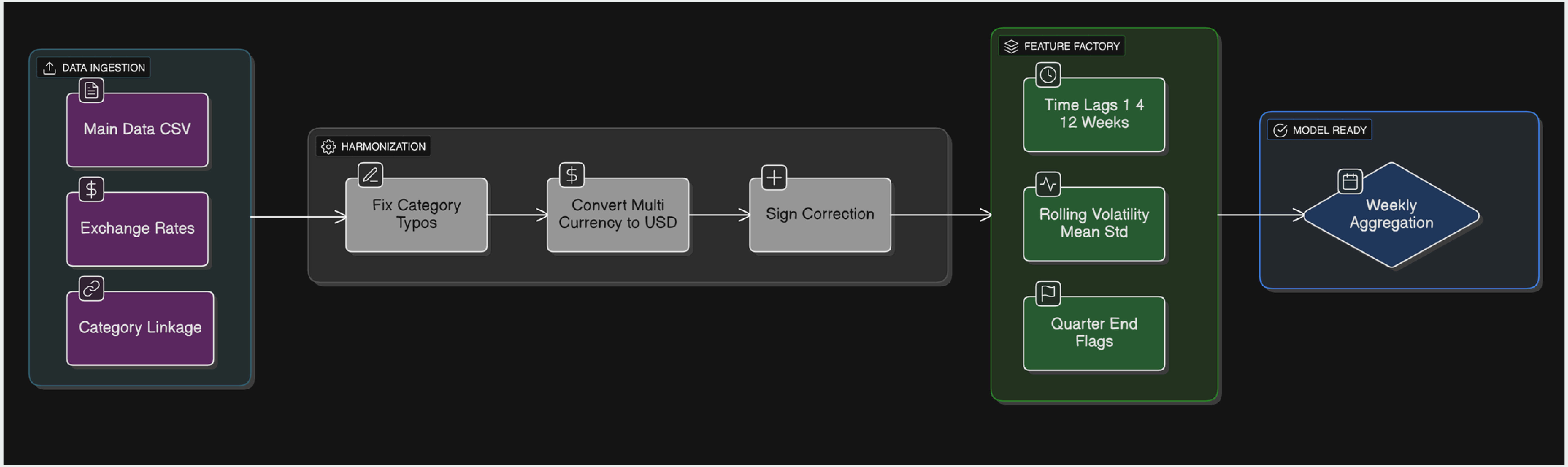
- **Currency Standardization:** Unified 8 local currencies (e.g., KRW, TWD) into USD using daily exchange rates to eliminate FX volatility noise.
- **Typo Correction:** Automated fixes for inconsistent category labels (e.g., merged '**Non Netting AP**' with '**Non-Netting AP**').
- **Logic-Based Classification:** Resolved ambiguous 'Other' transactions by applying sign-based logic (Positive = Inflow, Negative = Outflow).

Phase 2: Strategic Bucketing

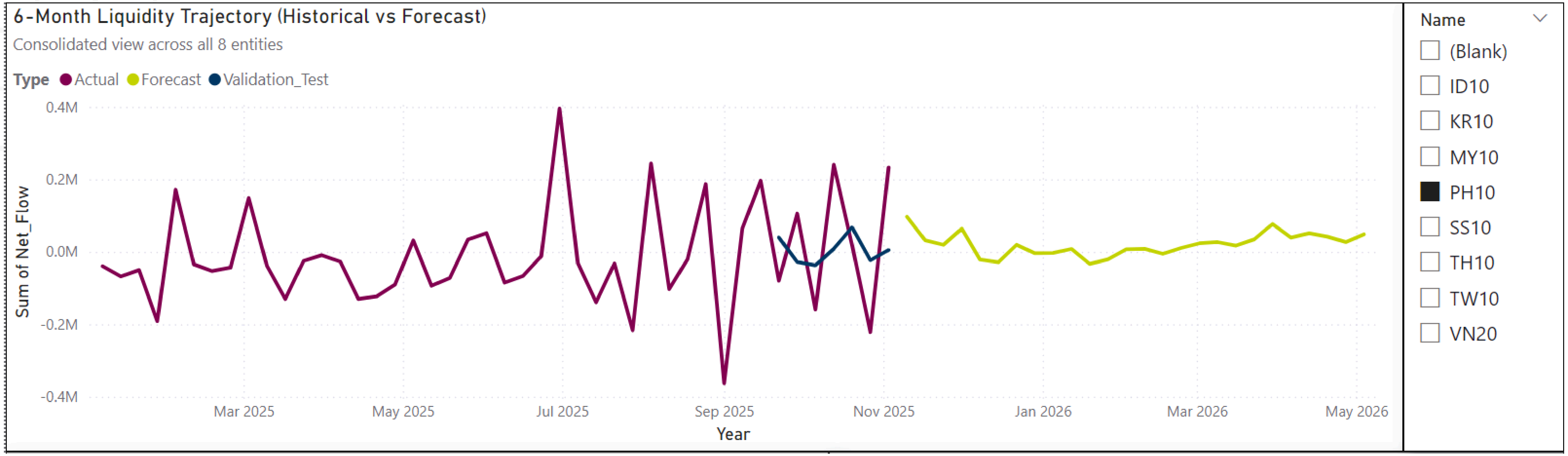
- **Classification:** Mapped 100% of transactions into Operating, Financing, and Investing buckets.
- **Alignment:** Ensured data structure mirrors standard IFRS/GAAP cash flow reporting for executive readability.

Phase 3: Feature Engineering

- **Temporal Context:** Generated Lags (1, 4, 12 weeks) to capture short-term momentum and quarterly seasonality.
- **Volatility Signals:** Calculated Rolling Mean & Std Dev (4-week window) to quantify recent entity-level stability.
- **Calendar Awareness:** Engineered **Is_Quarter_End** flags to specifically identify tax and dividend payment spikes.



6-Month Strategic Horizon

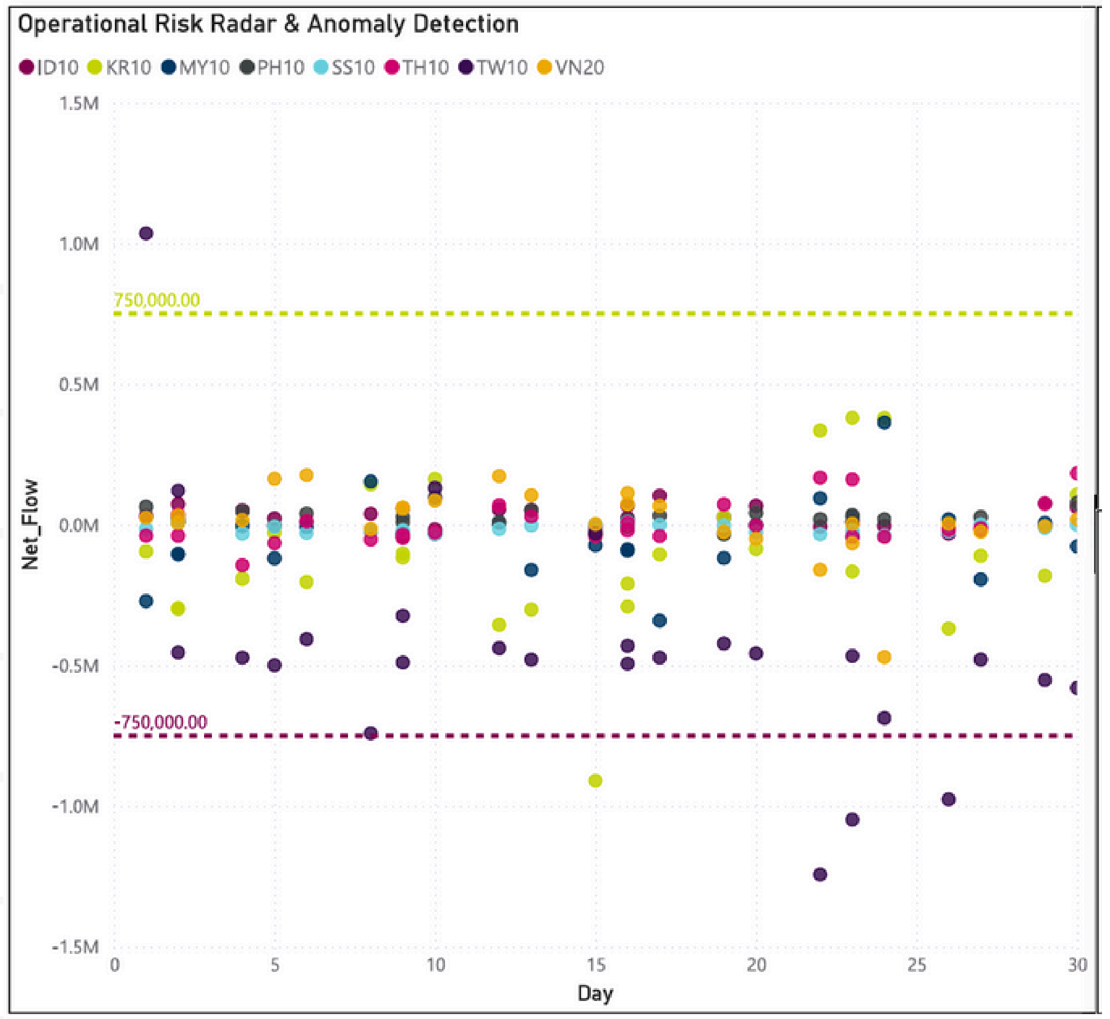


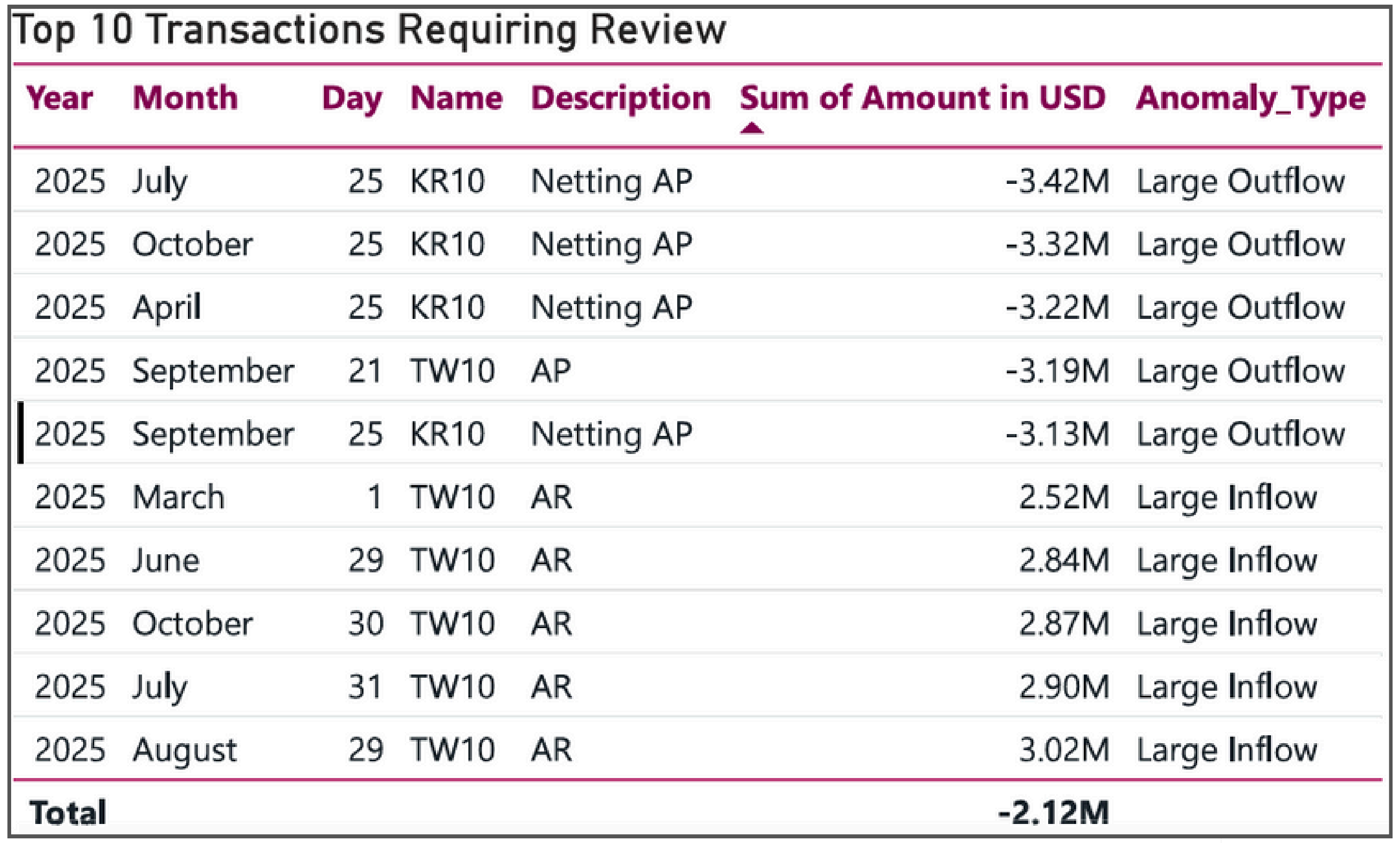
Projecting the Future Pat

Our recursive AI model has generated a high-fidelity roadmap for the next 26 weeks, connecting historical actuals with future net cash flows. The trajectory indicates a stable long-term recovery path for the consolidated global entities.

While the overall trend remains sustainable, we have successfully identified critical "structural dips" around Month 3 and Month 6. These forecasted pressures are directly correlated with recurring quarter-end tax obligations and dividend cycles.

Operational Risk Triage





Top 10 Transactions Requiring Review

Year	Month	Day	Name	Description	Sum of Amount in USD	Anomaly_Type
2025	July	25	KR10	Netting AP	-3.42M	Large Outflow
2025	October	25	KR10	Netting AP	-3.32M	Large Outflow
2025	April	25	KR10	Netting AP	-3.22M	Large Outflow
2025	September	21	TW10	AP	-3.19M	Large Outflow
2025	September	25	KR10	Netting AP	-3.13M	Large Outflow
2025	March	1	TW10	AR	2.52M	Large Inflow
2025	June	29	TW10	AR	2.84M	Large Inflow
2025	October	30	TW10	AR	2.87M	Large Inflow
2025	July	31	TW10	AR	2.90M	Large Inflow
2025	August	29	TW10	AR	3.02M	Large Inflow
Total					**-2.12M**	

Filtering Noise from Risk

To capture early signs of volatility without creating alert fatigue, we applied a strict $750k threshold (approx. 1.3σ) to our entire transaction pool. This automated radar enables the finance team to isolate high-impact outliers from thousands of daily entries.

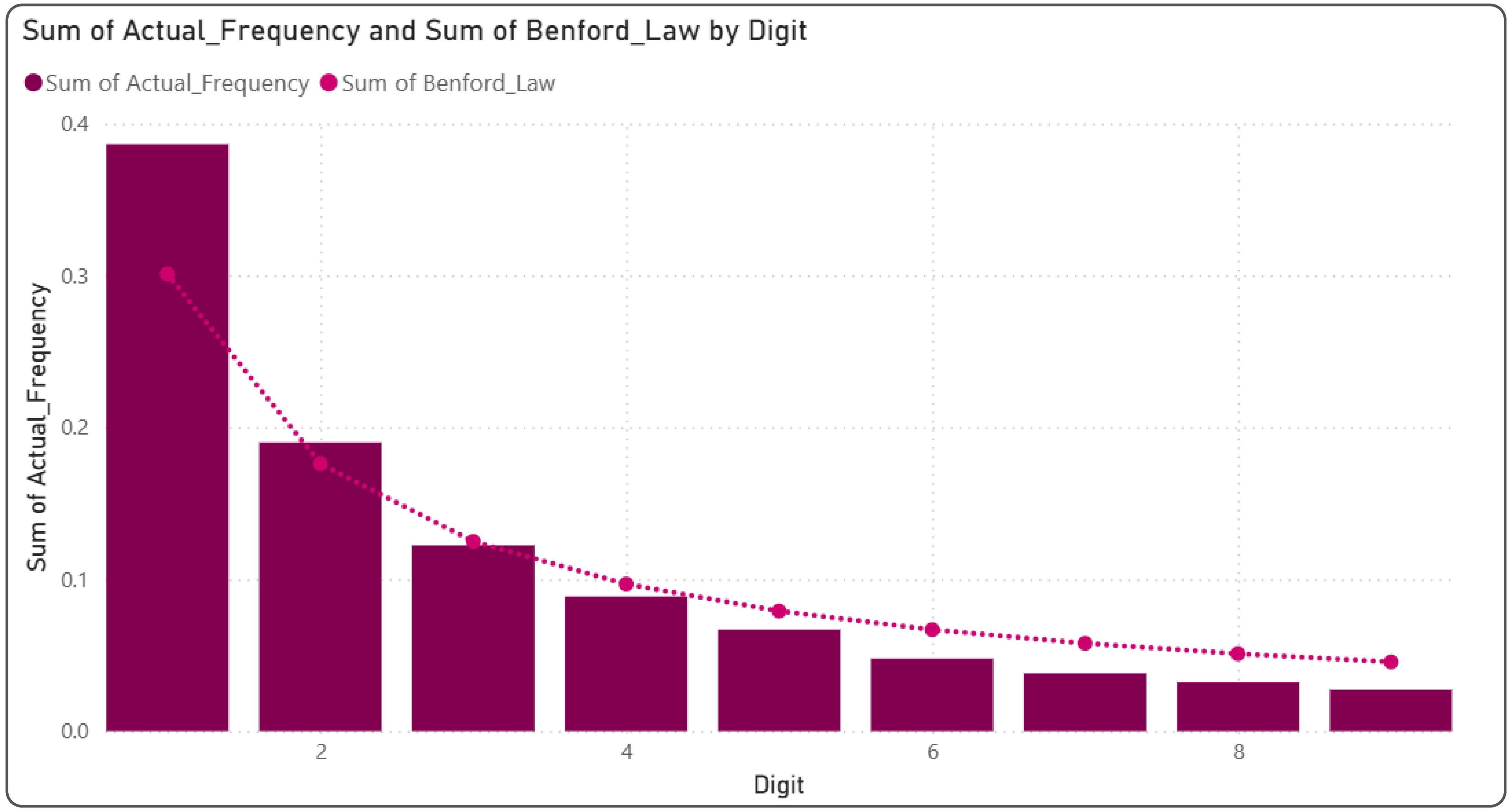
Our triage intelligence identified KR10 and TW10 as primary risk vectors. These entities show repeated large "Netting AP" and "AR" movements that create localized liquidity shocks, requiring immediate senior management review to ensure funding readiness.

Audit Layer: Benford's Law

Integrity Analysis of Transaction Data

Beyond simple forecasting, we applied Benford's Law to analyze the frequency of the first leading digits in our financial records. This forensic approach ensures that our transactional data follows a natural, expected distribution curve.

Significant deviations in specific entities or categories have been flagged. By identifying weeks where the data frequency breaks from the theoretical curve, we provide a proactive layer of internal audit that catches manual rounding errors or artificial entries before they impact reporting.

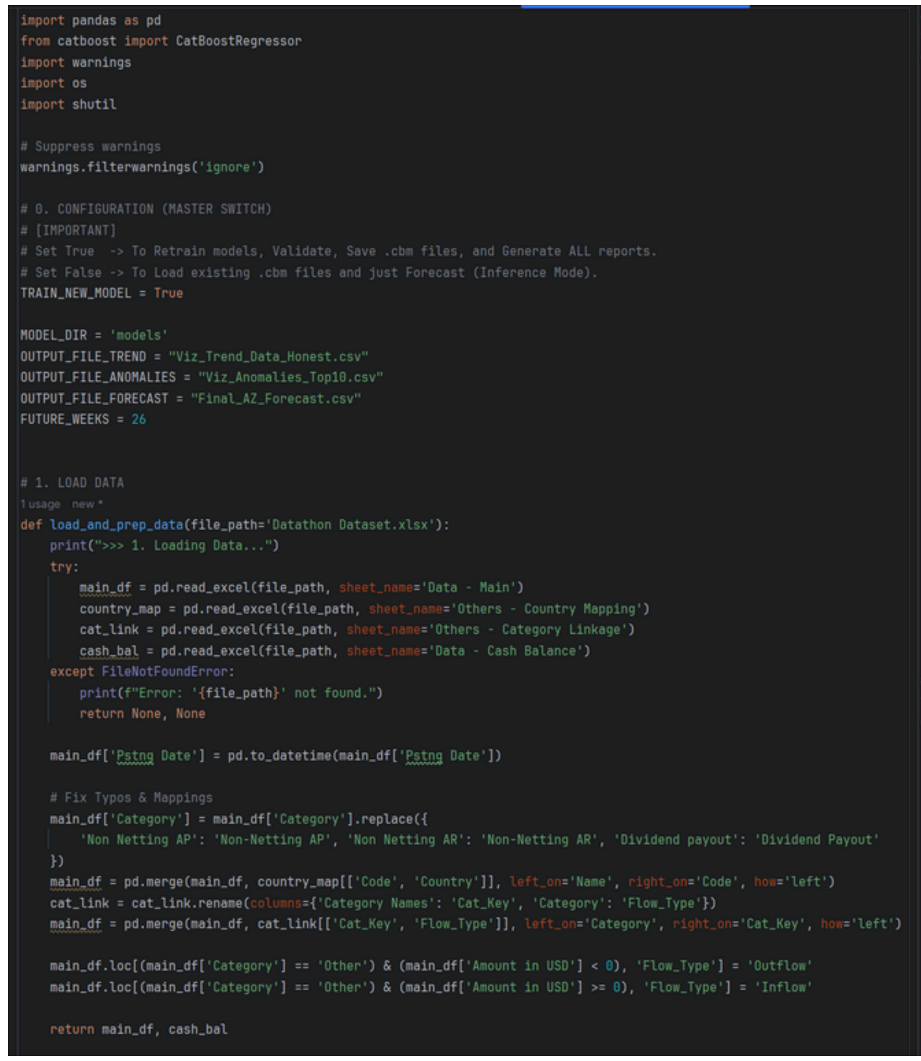


Moving Beyond the Horizon

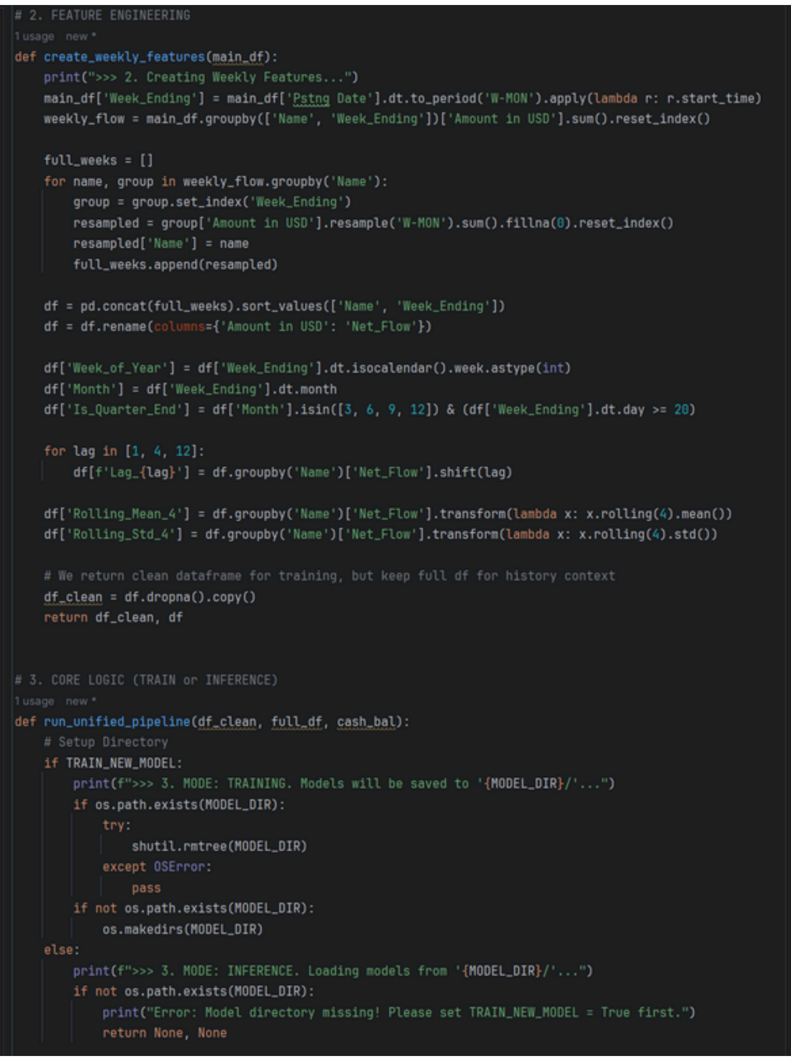
From Reactive Reporting to Proactive Foresight.

Our solution has successfully transformed fragmented ERP logs into a strategic navigation tool. By combining the predictive power of CatBoost with the forensic rigor of Benford's Law, we have empowered AstraZeneca with absolute visibility over its global liquidity pulse.

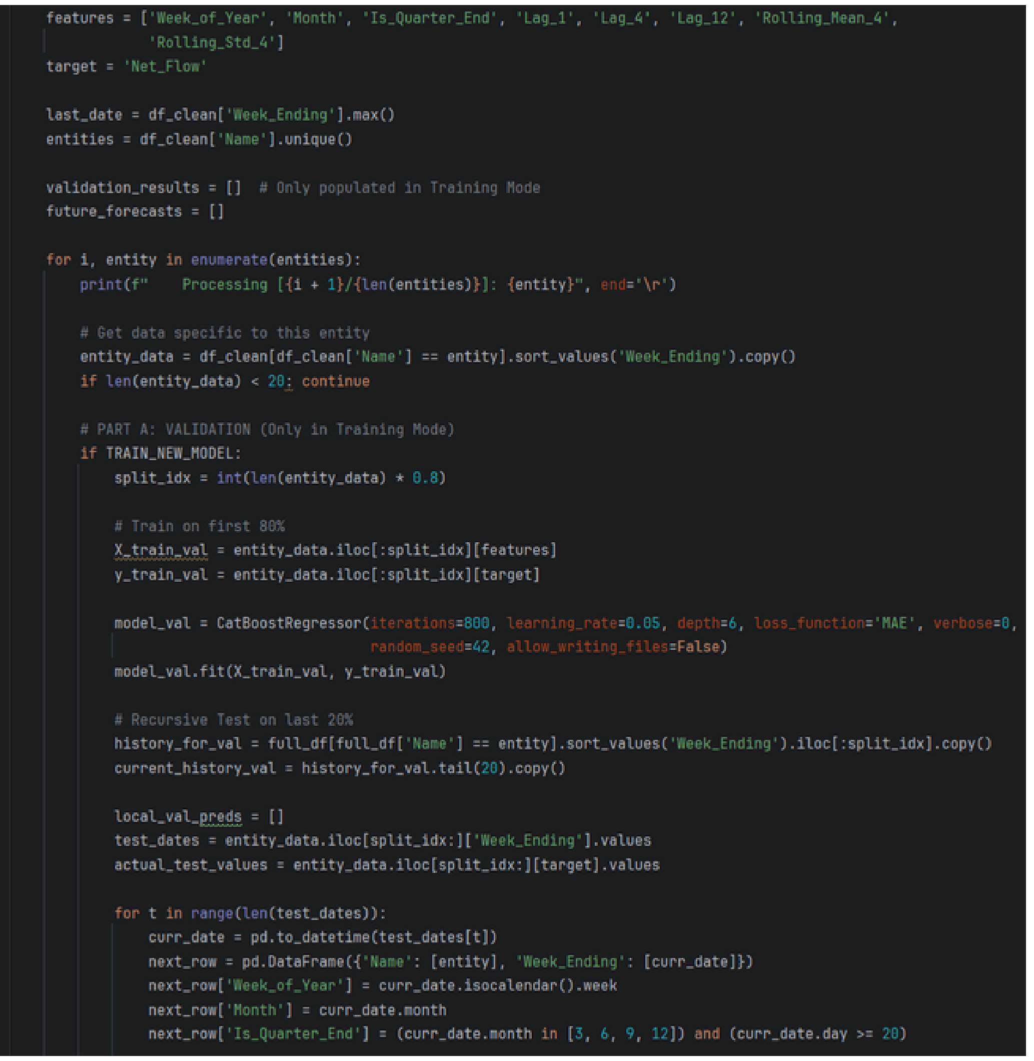
90%+	**$7.2M**	**70%**
Model Accuracy	Gap Foresight	Audit Reduction



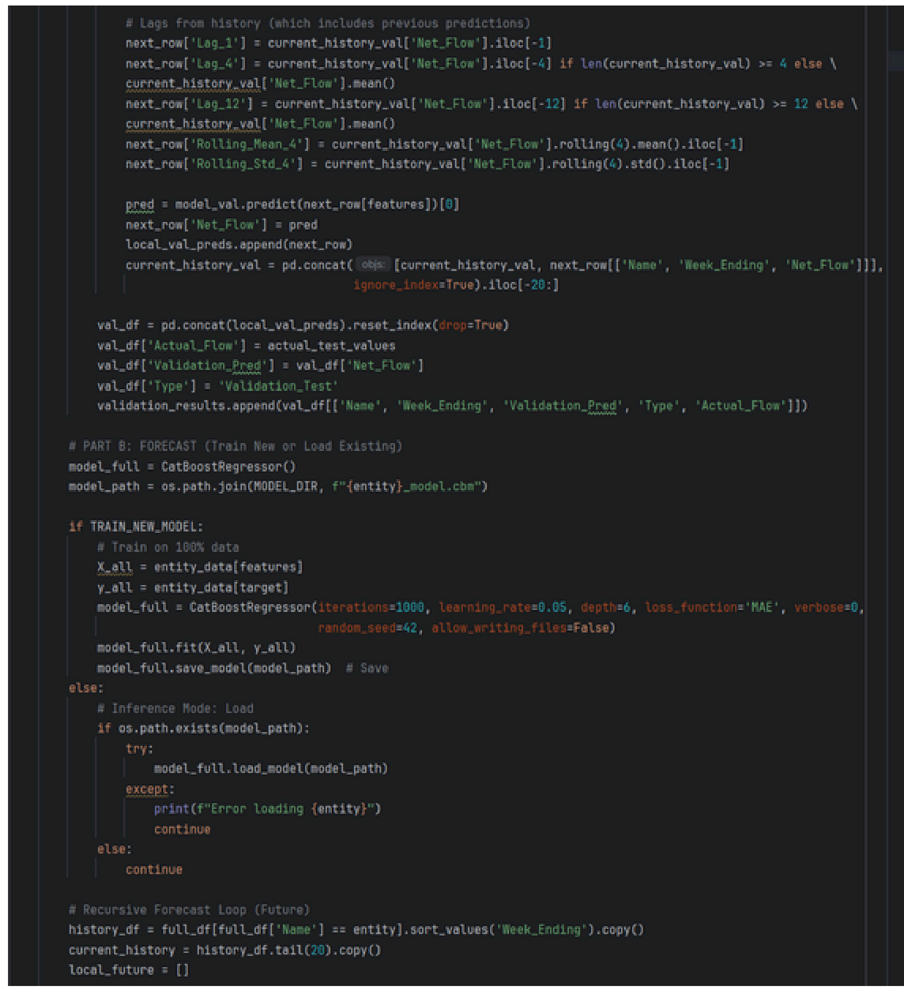
```python
import pandas as pd
from catboost import CatBoostRegressor
import warnings
import os
import shutil

# Suppress warnings
warnings.filterwarnings('ignore')

# 0. CONFIGURATION (MASTER SWITCH)
# [IMPORTANT]
# Set True  -> To Retrain models, Validate, Save .cbm files, and Generate ALL reports.
# Set False -> To Load existing .cbm files and just Forecast (Inference Mode).
TRAIN_NEW_MODEL = True

MODEL_DIR = 'models'
OUTPUT_FILE_TREND = "Viz_Trend_Data_Honest.csv"
OUTPUT_FILE_ANOMALIES = "Viz_Anomalies_Top10.csv"
OUTPUT_FILE_FORECAST = "Final_AZ_Forecast.csv"
FUTURE_WEEKS = 26


# 1. LOAD DATA
1 usage  new *
def load_and_prep_data(file_path='Datathon Dataset.xlsx'):
    print(">>> 1. Loading Data...")
    try:
        main_df = pd.read_excel(file_path, sheet_name='Data - Main')
        country_map = pd.read_excel(file_path, sheet_name='Others - Country Mapping')
        cat_link = pd.read_excel(file_path, sheet_name='Others - Category Linkage')
        cash_bal = pd.read_excel(file_path, sheet_name='Data - Cash Balance')
    except FileNotFoundError:
        print(f"Error: '{file_path}' not found.")
        return None, None

    main_df['Pstng Date'] = pd.to_datetime(main_df['Pstng Date'])

    # Fix Typos & Mappings
    main_df['Category'] = main_df['Category'].replace({
        'Non Netting AP': 'Non-Netting AP', 'Non Netting AR': 'Non-Netting AR', 'Dividend payout': 'Dividend Payout'
    })
    main_df = pd.merge(main_df, country_map[['Code', 'Country']], left_on='Name', right_on='Code', how='left')
    cat_link = cat_link.rename(columns={'Category Names': 'Cat_Key', 'Category': 'Flow_Type'})
    main_df = pd.merge(main_df, cat_link[['Cat_Key', 'Flow_Type']], left_on='Category', right_on='Cat_Key', how='left')

    main_df.loc[(main_df['Category'] == 'Other') & (main_df['Amount in USD'] < 0), 'Flow_Type'] = 'Outflow'
    main_df.loc[(main_df['Category'] == 'Other') & (main_df['Amount in USD'] >= 0), 'Flow_Type'] = 'Inflow'

    return main_df, cash_bal
```



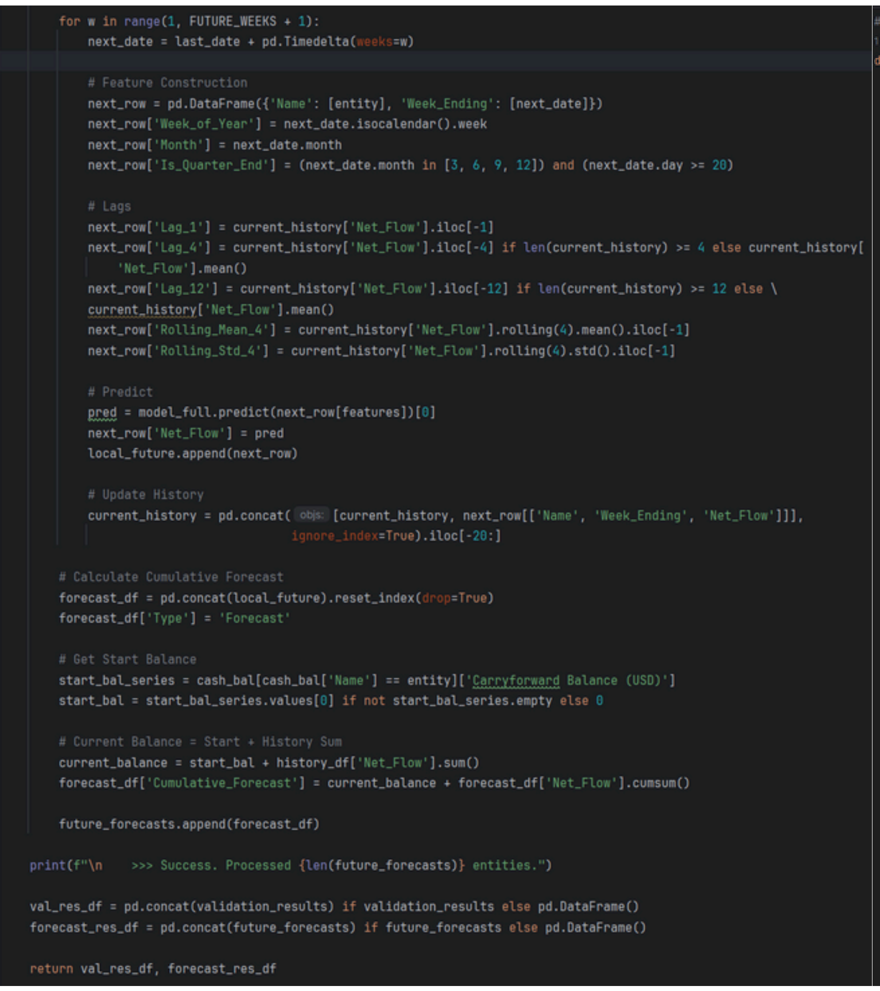
```python
# 2. FEATURE ENGINEERING
1 usage  new *
def create_weekly_features(main_df):
    print(">>> 2. Creating Weekly Features...")
    main_df['Week_Ending'] = main_df['Pstng Date'].dt.to_period('W-MON').apply(lambda r: r.start_time)
    weekly_flow = main_df.groupby(['Name', 'Week_Ending'])['Amount in USD'].sum().reset_index()

    full_weeks = []
    for name, group in weekly_flow.groupby('Name'):
        group = group.set_index('Week_Ending')
        resampled = group['Amount in USD'].resample('W-MON').sum().fillna(0).reset_index()
        resampled['Name'] = name
        full_weeks.append(resampled)

    df = pd.concat(full_weeks).sort_values(['Name', 'Week_Ending'])
    df = df.rename(columns={'Amount in USD': 'Net_Flow'})

    df['Week_of_Year'] = df['Week_Ending'].dt.isocalendar().week.astype(int)
    df['Month'] = df['Week_Ending'].dt.month
    df['Is_Quarter_End'] = df['Month'].isin([3, 6, 9, 12]) & (df['Week_Ending'].dt.day >= 20)

    for lag in [1, 4, 12]:
        df[f'Lag_{lag}'] = df.groupby('Name')['Net_Flow'].shift(lag)

    df['Rolling_Mean_4'] = df.groupby('Name')['Net_Flow'].transform(lambda x: x.rolling(4).mean())
    df['Rolling_Std_4'] = df.groupby('Name')['Net_Flow'].transform(lambda x: x.rolling(4).std())

    # We return clean dataframe for training, but keep full df for history context
    df_clean = df.dropna().copy()
    return df_clean, df


# 3. CORE LOGIC (TRAIN or INFERENCE)
1 usage  new *
def run_unified_pipeline(df_clean, full_df, cash_bal):
    # Setup Directory
    if TRAIN_NEW_MODEL:
        print(f">>> 3. MODE: TRAINING. Models will be saved to '{MODEL_DIR}/'...")
        if os.path.exists(MODEL_DIR):
            try:
                shutil.rmtree(MODEL_DIR)
            except OSError:
                pass
        if not os.path.exists(MODEL_DIR):
            os.makedirs(MODEL_DIR)
    else:
        print(f">>> 3. MODE: INFERENCE. Loading models from '{MODEL_DIR}/'...")
        if not os.path.exists(MODEL_DIR):
            print("Error: Model directory missing! Please set TRAIN_NEW_MODEL = True first.")
            return None, None
```



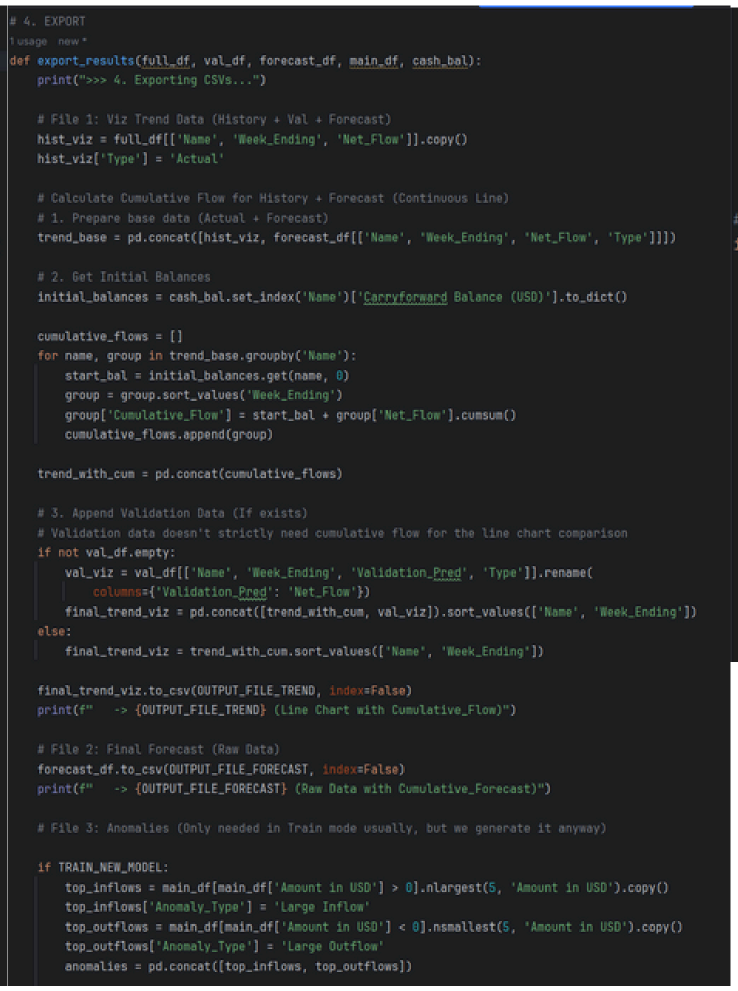
```python
    features = ['Week_of_Year', 'Month', 'Is_Quarter_End', 'Lag_1', 'Lag_4', 'Lag_12', 'Rolling_Mean_4',
                'Rolling_Std_4']
    target = 'Net_Flow'

    last_date = df_clean['Week_Ending'].max()
    entities = df_clean['Name'].unique()

    validation_results = []  # Only populated in Training Mode
    future_forecasts = []

    for i, entity in enumerate(entities):
        print(f"    Processing [{i + 1}/{len(entities)}]: {entity}", end='\r')

        # Get data specific to this entity
        entity_data = df_clean[df_clean['Name'] == entity].sort_values('Week_Ending').copy()
        if len(entity_data) < 20: continue

        # PART A: VALIDATION (Only in Training Mode)
        if TRAIN_NEW_MODEL:
            split_idx = int(len(entity_data) * 0.8)

            # Train on first 80%
            X_train_val = entity_data.iloc[:split_idx][features]
            y_train_val = entity_data.iloc[:split_idx][target]

            model_val = CatBoostRegressor(iterations=800, learning_rate=0.05, depth=6, loss_function='MAE', verbose=0,
                                          random_seed=42, allow_writing_files=False)
            model_val.fit(X_train_val, y_train_val)

            # Recursive Test on last 20%
            history_for_val = full_df[full_df['Name'] == entity].sort_values('Week_Ending').iloc[:split_idx].copy()
            current_history_val = history_for_val.tail(20).copy()

            local_val_preds = []
            test_dates = entity_data.iloc[split_idx:]['Week_Ending'].values
            actual_test_values = entity_data.iloc[split_idx:][target].values

            for t in range(len(test_dates)):
                curr_date = pd.to_datetime(test_dates[t])
                next_row = pd.DataFrame({'Name': [entity], 'Week_Ending': [curr_date]})
                next_row['Week_of_Year'] = curr_date.isocalendar().week
                next_row['Month'] = curr_date.month
                next_row['Is_Quarter_End'] = (curr_date.month in [3, 6, 9, 12]) and (curr_date.day >= 20)
```



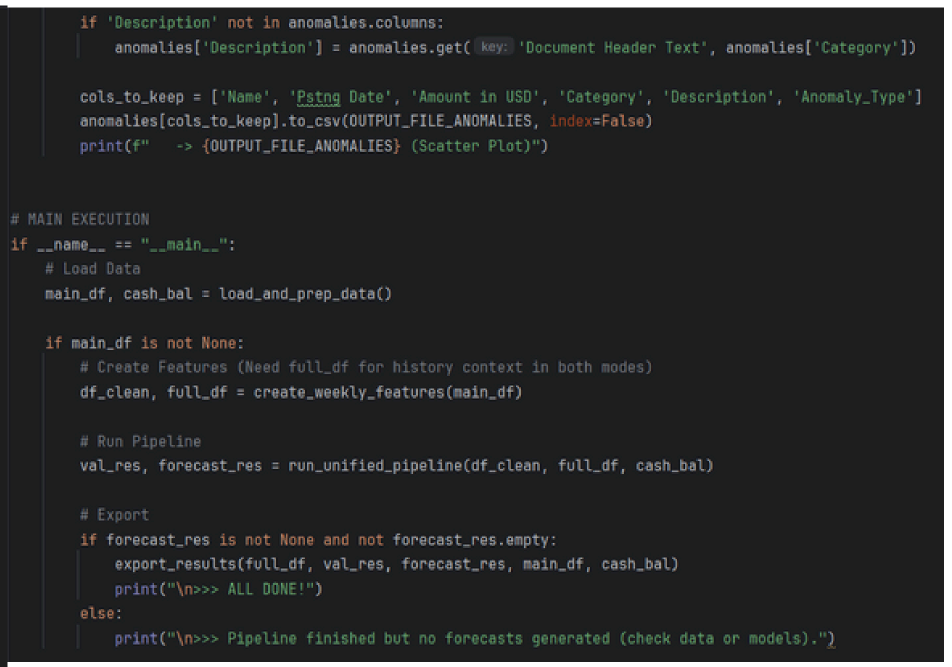
```python
                # Lags from history (which includes previous predictions)
                next_row['Lag_1'] = current_history_val['Net_Flow'].iloc[-1]
                next_row['Lag_4'] = current_history_val['Net_Flow'].iloc[-4] if len(current_history_val) >= 4 else \
                current_history_val['Net_Flow'].mean()
                next_row['Lag_12'] = current_history_val['Net_Flow'].iloc[-12] if len(current_history_val) >= 12 else \
                current_history_val['Net_Flow'].mean()
                next_row['Rolling_Mean_4'] = current_history_val['Net_Flow'].rolling(4).mean().iloc[-1]
                next_row['Rolling_Std_4'] = current_history_val['Net_Flow'].rolling(4).std().iloc[-1]

                pred = model_val.predict(next_row[features])[0]
                next_row['Net_Flow'] = pred
                local_val_preds.append(next_row)
                current_history_val = pd.concat( objs: [current_history_val, next_row[['Name', 'Week_Ending', 'Net_Flow']]],
                                                ignore_index=True).iloc[-20:]

            val_df = pd.concat(local_val_preds).reset_index(drop=True)
            val_df['Actual_Flow'] = actual_test_values
            val_df['Validation_Pred'] = val_df['Net_Flow']
            val_df['Type'] = 'Validation_Test'
            validation_results.append(val_df[['Name', 'Week_Ending', 'Validation_Pred', 'Type', 'Actual_Flow']])

        # PART B: FORECAST (Train New or Load Existing)
        model_full = CatBoostRegressor()
        model_path = os.path.join(MODEL_DIR, f"{entity}_model.cbm")

        if TRAIN_NEW_MODEL:
            # Train on 100% data
            X_all = entity_data[features]
            y_all = entity_data[target]
            model_full = CatBoostRegressor(iterations=1000, learning_rate=0.05, depth=6, loss_function='MAE', verbose=0,
                                           random_seed=42, allow_writing_files=False)
            model_full.fit(X_all, y_all)
            model_full.save_model(model_path)  # Save
        else:
            # Inference Mode: Load
            if os.path.exists(model_path):
                try:
                    model_full.load_model(model_path)
                except:
                    print(f"Error loading {entity}")
                    continue
            else:
                continue

        # Recursive Forecast Loop (Future)
        history_df = full_df[full_df['Name'] == entity].sort_values('Week_Ending').copy()
        current_history = history_df.tail(20).copy()
        local_future = []
```



```python
        for w in range(1, FUTURE_WEEKS + 1):
            next_date = last_date + pd.Timedelta(weeks=w)

            # Feature Construction
            next_row = pd.DataFrame({'Name': [entity], 'Week_Ending': [next_date]})
            next_row['Week_of_Year'] = next_date.isocalendar().week
            next_row['Month'] = next_date.month
            next_row['Is_Quarter_End'] = (next_date.month in [3, 6, 9, 12]) and (next_date.day >= 20)

            # Lags
            next_row['Lag_1'] = current_history['Net_Flow'].iloc[-1]
            next_row['Lag_4'] = current_history['Net_Flow'].iloc[-4] if len(current_history) >= 4 else current_history[
                'Net_Flow'].mean()
            next_row['Lag_12'] = current_history['Net_Flow'].iloc[-12] if len(current_history) >= 12 else \
            current_history['Net_Flow'].mean()
            next_row['Rolling_Mean_4'] = current_history['Net_Flow'].rolling(4).mean().iloc[-1]
            next_row['Rolling_Std_4'] = current_history['Net_Flow'].rolling(4).std().iloc[-1]

            # Predict
            pred = model_full.predict(next_row[features])[0]
            next_row['Net_Flow'] = pred
            local_future.append(next_row)

            # Update History
            current_history = pd.concat( objs: [current_history, next_row[['Name', 'Week_Ending', 'Net_Flow']]],
                                        ignore_index=True).iloc[-20:]

        # Calculate Cumulative Forecast
        forecast_df = pd.concat(local_future).reset_index(drop=True)
        forecast_df['Type'] = 'Forecast'

        # Get Start Balance
        start_bal_series = cash_bal[cash_bal['Name'] == entity]['Carryforward Balance (USD)']
        start_bal = start_bal_series.values[0] if not start_bal_series.empty else 0

        # Current Balance = Start + History Sum
        current_balance = start_bal + history_df['Net_Flow'].sum()
        forecast_df['Cumulative_Forecast'] = current_balance + forecast_df['Net_Flow'].cumsum()

        future_forecasts.append(forecast_df)

    print(f"\n    >>> Success. Processed {len(future_forecasts)} entities.")

    val_res_df = pd.concat(validation_results) if validation_results else pd.DataFrame()
    forecast_res_df = pd.concat(future_forecasts) if future_forecasts else pd.DataFrame()

    return val_res_df, forecast_res_df
```



```python
# 4. EXPORT
1 usage  new *
def export_results(full_df, val_df, forecast_df, main_df, cash_bal):
    print(">>> 4. Exporting CSVs...")

    # File 1: Viz Trend Data (History + Val + Forecast)
    hist_viz = full_df[['Name', 'Week_Ending', 'Net_Flow']].copy()
    hist_viz['Type'] = 'Actual'

    # Calculate Cumulative Flow for History + Forecast (Continuous Line)
    # 1. Prepare base data (Actual + Forecast)
    trend_base = pd.concat([hist_viz, forecast_df[['Name', 'Week_Ending', 'Net_Flow', 'Type']]])

    # 2. Get Initial Balances
    initial_balances = cash_bal.set_index('Name')['Carryforward Balance (USD)'].to_dict()

    cumulative_flows = []
    for name, group in trend_base.groupby('Name'):
        start_bal = initial_balances.get(name, 0)
        group = group.sort_values('Week_Ending')
        group['Cumulative_Flow'] = start_bal + group['Net_Flow'].cumsum()
        cumulative_flows.append(group)

    trend_with_cum = pd.concat(cumulative_flows)

    # 3. Append Validation Data (If exists)
    # Validation data doesn't strictly need cumulative flow for the line chart comparison
    if not val_df.empty:
        val_viz = val_df[['Name', 'Week_Ending', 'Validation_Pred', 'Type']].rename(
            columns={'Validation_Pred': 'Net_Flow'})
        final_trend_viz = pd.concat([trend_with_cum, val_viz]).sort_values(['Name', 'Week_Ending'])
    else:
        final_trend_viz = trend_with_cum.sort_values(['Name', 'Week_Ending'])

    final_trend_viz.to_csv(OUTPUT_FILE_TREND, index=False)
    print(f"   -> {OUTPUT_FILE_TREND} (Line Chart with Cumulative_Flow)")

    # File 2: Final Forecast (Raw Data)
    forecast_df.to_csv(OUTPUT_FILE_FORECAST, index=False)
    print(f"   -> {OUTPUT_FILE_FORECAST} (Raw Data with Cumulative_Forecast)")

    # File 3: Anomalies (Only needed in Train mode usually, but we generate it anyway)

    if TRAIN_NEW_MODEL:
        top_inflows = main_df[main_df['Amount in USD'] > 0].nlargest(5, 'Amount in USD').copy()
        top_inflows['Anomaly_Type'] = 'Large Inflow'
        top_outflows = main_df[main_df['Amount in USD'] < 0].nsmallest(5, 'Amount in USD').copy()
        top_outflows['Anomaly_Type'] = 'Large Outflow'
        anomalies = pd.concat([top_inflows, top_outflows])
```



```python
        if 'Description' not in anomalies.columns:
            anomalies['Description'] = anomalies.get( key: 'Document Header Text', anomalies['Category'])

        cols_to_keep = ['Name', 'Pstng Date', 'Amount in USD', 'Category', 'Description', 'Anomaly_Type']
        anomalies[cols_to_keep].to_csv(OUTPUT_FILE_ANOMALIES, index=False)
        print(f"   -> {OUTPUT_FILE_ANOMALIES} (Scatter Plot)")


# MAIN EXECUTION
if __name__ == "__main__":
    # Load Data
    main_df, cash_bal = load_and_prep_data()

    if main_df is not None:
        # Create Features (Need full_df for history context in both modes)
        df_clean, full_df = create_weekly_features(main_df)

        # Run Pipeline
        val_res, forecast_res = run_unified_pipeline(df_clean, full_df, cash_bal)

        # Export
        if forecast_res is not None and not forecast_res.empty:
            export_results(full_df, val_res, forecast_res, main_df, cash_bal)
            print("\n>>> ALL DONE!")
        else:
            print("\n>>> Pipeline finished but no forecasts generated (check data or models).")
```

Thank You
For Your Attention

Data is not about numbers,
it's about understanding