1. Importing Libraries

```
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import LabelEncoder, RobustScaler
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import logging
import requests
import openai
from jira import JIRA
import os
import sys
```

- **`pandas`**: For data manipulation (e.g., loading CSV, handling dataframes).
- **`numpy`**: For numerical operations, particularly useful in handling arrays and matrices.
- **`sklearn.ensemble.IsolationForest`**: A machine learning model used for anomaly detection based on isolation forest technique.
- **`sklearn.preprocessing.LabelEncoder, RobustScaler`**: Used for encoding categorical variables and scaling data to handle outliers.
- **`smtplib`**: For sending emails.
- **`email.mime.text` and `email.mime.multipart`**: For structuring and formatting emails.
- **`logging`**: To log the actions and errors for debugging and tracking.
- **`requests`**: To send HTTP requests, used for interacting with external APIs like Agentic AI.
- **`openai`**: For leveraging OpenAI’s language models to generate anomaly explanations.

- **`jira`**: For interacting with Jira API to create tickets for detected anomalies.
- **`os`** **and** **`sys`**: For handling system paths and error handling.

2. Logging Setup

```
logging.basicConfig(level=logging.INFO)
```

- This line sets up logging to capture and display messages of level `INFO` or higher, helping in tracking the progress and debugging errors during the execution.

3. API and Configuration Setup

```
openai.api_key = "your-openai-api-key"
jira_url = "https://your-domain.atlassian.net"
jira_user = "your-email@example.com"
jira_token = "your-jira-api-token"
jira = JIRA(jira_url, basic_auth=(jira_user, jira_token))
AGENTIC_API_KEY = "your-agentic-api-key"
```

- The **OpenAI API key** is used to generate anomaly explanations.
- The **JIRA setup** is for managing tasks created during anomaly detection.
- **Agentic AI API key** is used for creating tasks in the Agentic AI system for automated task management.

4. Data Loading and Preprocessing

Function: **`load_data_tool(file_path)`**

```
def load_data_tool(file_path):
    if not os.path.exists(file_path):
        logging.error(f"File {file_path} not found.")
        sys.exit(1)

    try:
        df = pd.read_csv(file_path, dtype={"Account": "category",
"AU": "category", "Company": "category"})
    except Exception as e:
        logging.error(f"Error loading file {file_path}: {str(e)}")
        sys.exit(1)

    required_cols = ['Asofdate', 'Company', 'Account', 'AU', 'Match
Status', 'GL Balance', 'IHub balance', 'Balance difference']
    missing_cols = [col for col in required_cols if col not in
df.columns]
    if missing_cols:
        logging.error(f"Missing columns {missing_cols}")
        sys.exit(1)

    df.dropna(subset=required_cols, inplace=True)
```

- **File existence check**: Verifies if the file exists at the given path.
- **Loading data**: The CSV is read into a pandas DataFrame. The `dtype` option ensures that columns like `Account`, `AU`, and `Company` are treated as categorical.
- **Column validation**: Ensures the presence of necessary columns like `Asofdate`, `Company`, `Account`, etc. If any columns are missing, it exits the script.

Label Encoding

```
   le_account = LabelEncoder()
    le_au = LabelEncoder()
    le_company = LabelEncoder()

    df['Company'] =
le_company.fit_transform(df['Company'].astype(str))
    df['Account'] =
le_account.fit_transform(df['Account'].astype(str))
```

```
    df['AU'] = le_au.fit_transform(df['AU'].astype(str))
    df['Match Status'] = LabelEncoder().fit_transform(df['Match
Status'].astype(str))
    df['Asofdate'] = pd.to_datetime(df['Asofdate'], errors='coerce')
```

- **Label Encoding**: The categorical columns (`Company`, `Account`, `AU`, and `Match Status`) are transformed into numerical values using `LabelEncoder`.
- **Date conversion**: The `Asofdate` column is converted into a proper `datetime` format.

5. Feature Engineering

Function: `engineer_features_with_llm(df)`

```
def engineer_features_with_llm(df):
    for col in ['GL Balance', 'IHub balance', 'Balance difference']:
        df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0)
```

- **Data Cleaning**: Ensures that numerical columns (`GL Balance`, `IHub balance`, `Balance difference`) are correctly converted into numeric types. If conversion fails (e.g., due to non-numeric values), it fills those values with `0`.

```
   df = df.sort_values('Asofdate')
    grouped = df.groupby(['Company', 'Account', 'AU'],
group_keys=False)
```

- **Sorting and Grouping**: The data is sorted by `Asofdate` (date of the transaction) and grouped by `Company`, `Account`, and `AU`.

```
   df['Days_Since_Last'] =
grouped['Asofdate'].diff().dt.days.fillna(0)
```

```
    df['GL_Change'] = grouped['GL Balance'].diff().fillna(0)
    df['IHub_Change'] = grouped['IHub balance'].diff().fillna(0)
    df['Diff_Change'] = grouped['Balance difference'].diff().fillna(0)
    df['GL_Std'] = grouped['GL Balance'].transform(lambda x:
x.rolling(window=3, min_periods=1).std().fillna(0))
    df['IHub_Std'] = grouped['IHub balance'].transform(lambda x:
x.rolling(window=3, min_periods=1).std().fillna(0))
```

- **Feature Creation**: New features are created, such as:
    - `Days_Since_Last`: Days since the last recorded transaction for each group.
    - `GL_Change`, `IHub_Change`, `Diff_Change`: Changes in `GL Balance`, `IHub Balance`, and `Balance Difference` between consecutive transactions.
    - `GL_Std`, `IHub_Std`: Rolling standard deviation over the last 3 periods to measure the variability of `GL Balance` and `IHub balance`.

```
   features = ['Company', 'Account', 'AU', 'Days_Since_Last', 'GL
Balance', 'IHub balance', 'Balance difference',
                'GL_Change', 'IHub_Change', 'Diff_Change', 'GL_Std',
'IHub_Std']
    return df, features
```

- **Features for the Model**: Returns the transformed dataframe along with the list of selected features.

6. Contamination Estimation

Function: `estimate_contamination(df, feature='Balance difference', z_threshold=3)`

```
def estimate_contamination(df, feature='Balance difference',
z_threshold=3):
    mean_diff = df[feature].mean()
    std_diff = df[feature].std()
    extreme_values = (df[feature].abs() > (mean_diff + z_threshold *
std_diff)).sum()
    contamination = min(max(extreme_values / len(df), 0.01), 0.5)
    logging.info(f"Estimated contamination: {contamination:.4f}")
    return contamination
```

- **Contamination Estimation**: Based on the `Balance difference` column, it calculates the number of extreme values (outliers) based on a Z-score threshold (`z_threshold`). This helps estimate the proportion of the data that may be anomalous (contamination level).

7. Anomaly Detection (Isolation Forest)

Function: `train_model_tool(df, features)`

```
def train_model_tool(df, features):
    X = df[features]
    scaler = RobustScaler()
    X_scaled = scaler.fit_transform(X)
```

- **Data Scaling**: The data is scaled using `RobustScaler` to minimize the influence of outliers.

```
   contamination = estimate_contamination(df)
    model = IsolationForest(n_estimators=200,
contamination=contamination, random_state=42, n_jobs=-1)
    model.fit(X_scaled)
```

- **Isolation Forest**: The model is trained using the `IsolationForest` algorithm, which detects anomalies by isolating observations that differ significantly from the rest of the data.

```
   df['Anomaly_Score'] = model.decision_function(X_scaled)
    df['Anomaly'] = model.predict(X_scaled)
    df['Anomaly'] = df['Anomaly'].apply(lambda x: 1 if x == -1 else 0)
    return model, df
```

- **Anomaly Labeling**: The model assigns an anomaly score, and any data point with a score indicating anomaly (`-1` from `predict()`) is labeled as `1` (anomaly).

8. Generating Anomaly Explanations Using OpenAI

Function: `generate_enhanced_explanation(anomaly_data)`

```
def generate_enhanced_explanation(anomaly_data):
    prompt = f"""
    Anomaly Detected:
    - GL Balance: {anomaly_data['GL Balance']}
    - IHub Balance: {anomaly_data['IHub balance']}
    - Balance Difference: {anomaly_data['Balance difference']}
    - Days Since Last Transaction: {anomaly_data['Days_Since_Last']}
    - GL Balance Change: {anomaly_data['GL_Change']}
    - IHub Balance Change: {anomaly_data['IHub_Change']}
    - Difference Change: {anomaly_data['Diff_Change']}
    - GL Balance Standard Deviation: {anomaly_data['GL_Std']}
    - IHub Balance Standard Deviation: {anomaly_data['IHub_Std']}

    Please provide an in-depth explanation for this anomaly...
    """
    response = openai.Completion.create(
        model="text-davinci-003",
        prompt=prompt,
        max_tokens=300
    )
    explanation = response['choices'][0]['text'].strip()
    return explanation
```

- **OpenAI for Explanation**: Sends the anomaly data to OpenAI's GPT model and gets a detailed explanation of the anomaly based on the provided data points (GL Balance, IHub Balance, etc.).

9. Creating Tasks in Agentic AI and JIRA

Function: `create_agentic_task(anomaly)`

```
def create_agentic_task(anomaly):
    api_url = "https://api.agentic.ai/tasks"
    headers = {
        "Authorization": f"Bearer {AGENTIC_API_KEY}",
        "Content-Type": "application/json"
    }

    task_data = {
        "task_name": f"Investigate Anomaly: {anomaly['id']}",
        "description": anomaly['explanation'],
        "severity": "High",
        "due_date": "2023-04-01"
    }

    try:
        response = requests.post(api_url, json=task_data,
headers=headers)
        if response.status_code == 200:
            logging.info(f"Task created successfully in Agentic AI.")
        else:
            logging.error(f"Error creating task in Agentic AI:
{response.text}")
    except Exception as e:
        logging.error(f"Exception while creating task in Agentic AI:
{str(e)}")
```

- **Agentic AI Task Creation**: Creates a task in Agentic AI to investigate the detected anomaly, with relevant information such as severity and due date.

Function: **`create_jira_ticket(anomaly)`**

```
def create_jira_ticket(anomaly):
    summary = f"Anomaly Detected: {anomaly['id']}"
    description = f"Details of anomaly:
{anomaly['description']}\nSeverity: High"
    new_issue = jira.create_issue(
        project='YOUR_PROJECT_KEY',
        summary=summary,
        description=description,
        issuetype={'name': 'Task'}
    )
    logging.info(f"JIRA ticket created: {new_issue.key}")
```

- **JIRA Ticket Creation**: Generates a new task in Jira with the anomaly details.

10. Sending Email Notification

Function: **`send_email_tool(anomalies, recipient_email, le_account, le_au, le_company)`**

```
def send_email_tool(anomalies, recipient_email, le_account, le_au,
le_company):
    sender_email = "youremailid"
    sender_password = "your-app-password"

    if anomalies.empty:
        logging.info("No anomalies detected. Skipping email.")
        return
```

- **Email Setup**: Sends an email notification about detected anomalies.

```
   subject = "Anomalies Detected"
```

```
    body = anomalies.to_html()

    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = recipient_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'html'))
```

- **Formatted Email**: The anomalies are converted to an HTML format and attached to the email.

```
   with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
        server.login(sender_email, sender_password)
        server.sendmail(sender_email, recipient_email,
msg.as_string())
```

- **SMTP**: Sends the email through Gmail's SMTP server.

11. Feedback Loop and Model Retraining

Function: `feedback_loop_and_retrain(df, feedback_data, model, features)`

```
def feedback_loop_and_retrain(df, feedback_data, model, features):
    logging.info("Applying feedback to adjust anomaly detection
model...")
```

- **Feedback Processing**: Incorporates feedback from users to retrain the model. Positive feedback (anomalies are real) and negative feedback (false positives) are used to adjust the contamination level and retrain the model.

```
   contamination = estimate_contamination(df)
```

```
    model = IsolationForest(n_estimators=200,
contamination=contamination, random_state=42, n_jobs=-1)
    model.fit(df[features])
    logging.info("Model retrained with feedback.")
    return model
```

- **Retraining**: The model is retrained with updated contamination levels based on the feedback.

12. Main Execution

Function: `main(file_path, feedback_file_path)`

```
def main(file_path, feedback_file_path):
    df, le_account, le_au, le_company = load_data_tool(file_path)
    feedback_data = pd.read_csv(feedback_file_path)
```

- **Data Loading**: Loads the primary dataset and feedback data.

```
   df, features = engineer_features_with_llm(df)
    model, df = train_model_tool(df, features)
```

- **Feature Engineering and Model Training**: Prepares the data and trains the anomaly detection model.

```
   if feedback_data is not None and not feedback_data.empty:
        model = feedback_loop_and_retrain(df, feedback_data, model,
features)
```

- **Feedback Loop**: Retrains the model if feedback is provided.

```
   send_email_tool(df[df['Anomaly'] == 1], "recipient@example.com",
le_account, le_au, le_company)
```

- **Email Notification**: Sends an email about the detected anomalies.

13. Execution Example

```
file_path = "/path/to/your/file.csv"
feedback_file_path = "/path/to/your/feedback_file.csv"
main(file_path, feedback_file_path)
```

- **File Paths**: The main function is called with paths to the data and feedback files.

Summary of Key Components:

1. **Data Loading**: Loads, cleans, and preprocesses the data.
2. **Feature Engineering**: Creates new features to improve anomaly detection.
3. **Anomaly Detection**: Uses the Isolation Forest model to detect anomalies.
4. **LLM-based Explanations**: Generates detailed explanations for detected anomalies using OpenAI's GPT model.
5. **Task Management**: Creates tasks in Agentic AI and JIRA for further investigation.
6. **Email Notifications**: Sends emails about anomalies.
7. **Feedback Loop**: Allows for model retraining based on feedback, improving detection accuracy.