Python Notes:

Extracting Data from a PDF and Populate a Spreadsheet &

Extracting Data from a Spreadsheet and Populate a PDF

1. **Extracting Data from PDFs to Spreadsheets**
   1. Key Libraries:
      1. **PyPDF2**: Extracts text from PDFs.
      2. **pdfplumber**: Handles more complex layouts, including tables.
      3. **tabula-py**: Specifically designed for extracting tabular data from PDFs.
      4. **camelot**: Advanced for structured table extraction.
2. **Filling Out PDFs with Spreadsheet Data**
   1. Key Libraries:
      1. **fpdf**: Create PDFs programmatically.
      2. **PyPDF2**: Modify existing PDFs.
      3. **pdfrw**: Fill out PDF forms.
      4. **reportlab**: Advanced PDF generation.
3. **Automating the Workflow**
   1. Develop Master Script
      1. Reads and Extracts PDF data and Exports to Excel for analysis.
      2. Reads back from Excel to fill out required forms in PDFs.
      3. Logs actions in a separate tracking file for compliance audits.

**Real-Life Applications in Your Role:**

* Manifests: Extract data from e-Manifest PDF files into a structured format for reporting.
* Compliance Reports: Automate population of standardized report templates using Excel data.
* Regulatory Submissions: Convert scanned regulatory documents into analyzable formats.
* Audit Preparation: Extract data from inspection reports and create summaries.

**Status of Manifest for tomorrow’s meeting.**

Python Notes:

Extracting Data from a Spreadsheet and Populate a PDF

To automate filling out a PDF based on data in a spreadsheet, the key is mapping spreadsheet columns to specific fields in the PDF. This involves identifying form fields in the PDF and programmatically assigning data from the spreadsheet to those fields. Here’s how to do it step by step:

1. **Understand the Structure of the PDF**
   1. **Form Field Identification:**
   * PDFs with fillable fields (e.g., Adobe Acrobat forms) have pre-defined field names.
   * Use a tool like Adobe Acrobat Pro or a PDF editor to inspect the PDF and locate the exact names of the fields.
   * Open the PDF in Acrobat, switch to "Prepare Form" mode, and note the field names (e.g., "TSDF").
   * If the PDF is not a fillable form, consider creating one by converting static text areas into fillable fields.
   1. **Mapping Spreadsheet Columns to PDF Fields:**
      1. Create a mapping between the spreadsheet columns (e.g., "TSDF") and the PDF field names (e.g., "Field\_TSDF").
2. **Extract Data from a Spreadsheet**
   1. Read the Data using Pandas
3. **Populate a PDF with Data from the Spreadsheet**
   1. Use pdfrw or pypdf2 to fill the fields in a PDF.
      1. Install Required Libraries
      2. Fill Out the PDF

|  |
| --- |
| pip install pdfrw pandas |

|  |
| --- |
| from pdfrw import PdfReader, PdfWriter, PageMerge  # Input and output file paths  template\_pdf = "template.pdf" # The fillable PDF template  output\_pdf = "filled\_form.pdf"  spreadsheet\_path = "data.xlsx"  # Load data from the spreadsheet  data = pd.read\_excel(spreadsheet\_path)  # Function to update PDF fields  def fill\_pdf(template, output, data):  pdf = PdfReader(template)  # Go through each page to find annotations  for page in pdf.pages:  annotations = page.Annots  if annotations:  for annotation in annotations:  key = annotation.T[1:-1] # Extract field name (e.g., "TSDF")  if key in data.columns:  value = data[key][0] # Assuming one row of data for simplicity  annotation.update({  "/V": value, # Set the value  "/Ff": 1, # Mark as read-only  })  # Write to a new PDF file  PdfWriter(output, trailer=pdf).write()  print(f"PDF filled and saved to {output}")  # Example: Populate PDF using the first row of data  fill\_pdf(template\_pdf, output\_pdf, data) |

1. **Key Details to Consider**

* Matching Columns to PDF Fields
  + The column name in the spreadsheet must match the field name in the PDF exactly.
    - Spreadsheet column: TSDF
    - PDF field: /T('TSDF')
* Non-Fillable PDFs
  + If the PDF is not fillable, convert it into a fillable template.
  + Alternatively, use reportlab to create a new PDF with populated data overlayed.

1. **Automating for Multiple Rows**

If the spreadsheet contains multiple rows (e.g., one row per TSDF entry), generate separate PDFs for each row:

|  |
| --- |
| for index, row in data.iterrows():  output\_pdf = f"filled\_form\_{index}.pdf"  fill\_pdf(template\_pdf, output\_pdf, row) |

1. **Extracting Data from a PDF to Populate a Spreadsheet**

If you need to extract data from a PDF instead, use pdfplumber or PyPDF2.

|  |
| --- |
| import pdfplumber  pdf\_path = "filled\_form.pdf"  output\_spreadsheet = "extracted\_data.xlsx"  with pdfplumber.open(pdf\_path) as pdf:  for page in pdf.pages:  text = page.extract\_text()  print(text) # Parse text or save it into a spreadsheet |

1. **Mapping Strategy**

* Identify Field Names:
  + Use pdfrw to print all form field names from the template:
* Create a Mapping File:
  + Use a dictionary to map spreadsheet columns to PDF field names:

|  |  |
| --- | --- |
| from pdfrw import PdfReader  template\_pdf = "template.pdf"  pdf = PdfReader(template\_pdf)  for page in pdf.pages:  annotations = page.Annots  if annotations:  for annotation in annotations:  print(annotation.T) # Field names | field\_mapping = {  "TSDF": "TSDF\_Field",  "Waste Volume": "Waste\_Volume\_Field",  } |

**Key Cybersecurity Concerns**

**1. Data Sensitivity and Compliance**

* **Issue**: Waste manifests, regulatory forms, and compliance reports often include sensitive data (e.g., generator details, waste classifications).
* **Risk**: Unauthorized access or exposure of this data could lead to non-compliance with regulations like HIPAA (if medical waste is involved) or CCPA/GDPR for personal data.

**Mitigation**:

* Use **encryption** for all data at rest and in transit.
* Restrict access to sensitive files using **role-based access control (RBAC)**.

**2. Unencrypted PDF and Spreadsheet Data**

* **Issue**: PDFs and Excel files are often stored or transmitted without encryption, leaving them vulnerable to interception or unauthorized access.
* **Risk**: Attackers could extract sensitive information or tamper with critical data.

**Mitigation**:

* Encrypt PDF Files.
* Encrypt Excell Files

|  |
| --- |
| from PyPDF2 import PdfReader, PdfWriter  writer = PdfWriter()  reader = PdfReader("output.pdf")  for page in reader.pages:  writer.add\_page(page)  writer.encrypt(user\_pwd="userpassword", owner\_pwd="adminpassword", use\_128bit=True)  with open("encrypted\_output.pdf", "wb") as f:  writer.write(f) |

|  |
| --- |
| from openpyxl import load\_workbook  from openpyxl import Workbook  # Open workbook and save with password protection  workbook = load\_workbook("data.xlsx")  workbook.security.workbook\_password = "securepassword"  workbook.save("protected\_data.xlsx") |

**3. Data Transmission**

* **Issue**: Transferring files (e.g., PDFs or spreadsheets) over insecure channels like email or non-encrypted file shares.
* **Risk**: Files could be intercepted, exposing sensitive information.

**Mitigation**:

* Use secure file transfer protocols (e.g., **SFTP**, **HTTPS**, **FTPS**) for transmission.
* Avoid using public cloud storage or email for file sharing unless encrypted.

**4. Temporary File Vulnerabilities**

* **Issue**: Temporary files created during processing (e.g., intermediate data extracted from PDFs) may not be deleted securely.
* **Risk**: These files could be accessed by unauthorized users.

**Mitigation**:

* Use Python’s os module to securely delete temporary files:

|  |
| --- |
| import os  os.remove("temp\_file.txt") |

**5. Insecure Dependencies**

* **Issue**: Python libraries used for PDF and spreadsheet manipulation (e.g., PyPDF2, pandas) may have vulnerabilities.
* **Risk**: Exploits in third-party libraries could expose your system to attacks.

**Mitigation**:

* Regularly update Python libraries:
* Use vulnerability scanning tools (e.g., **Bandit**, **Snyk**) to check for insecure dependencies:

|  |  |
| --- | --- |
| pip install --upgrade PyPDF2 pandas | pip install bandit  bandit -r your\_script.py |

**6. Insider Threats**

* **Issue**: Employees or contractors with access to sensitive data could misuse it.
* **Risk**: Insider threats could compromise compliance or lead to data breaches.

**Mitigation**:

* Implement logging for all access and changes to sensitive files.
* Use least-privilege access policies to minimize risk.

**7. Data Integrity**

* **Issue**: Manipulated or corrupted data (e.g., tampered waste manifests) could lead to compliance violations or operational errors.
* **Risk**: Unauthorized modifications could go undetected.

**Mitigation**:

* Implement digital signatures for PDFs:
* Use hashing for data integrity verification.

|  |
| --- |
| from PyPDF2 import PdfWriter  writer = PdfWriter()  writer.add\_metadata({"/Author": "Environmental Compliance Specialist"})  with open("signed\_output.pdf", "wb") as f:  writer.write(f) |

|  |
| --- |
| import hashlib  with open("data.xlsx", "rb") as f:  data = f.read()  print(hashlib.sha256(data).hexdigest()) |

**8. Access Management**

* **Issue**: Shared passwords or poor password practices could lead to unauthorized access.
* **Risk**: Data or systems could be compromised if credentials are leaked.

**Mitigation**:

* Enforce strong password policies.
* Implement multi-factor authentication (MFA) for systems accessing sensitive data.

**Additional Security Best Practices**

1. **Secure Coding Practices**:
   * Use exception handling to prevent crashes that might expose sensitive data.
   * Avoid hardcoding sensitive information (e.g., passwords) in scripts.
2. **Audit and Monitoring**:
   * Monitor logs for suspicious activity.
   * Regularly audit data handling practices to identify potential vulnerabilities.
3. **Data Retention Policies**:
   * Define policies for how long sensitive files (PDFs, spreadsheets) are stored.
   * Automatically delete files after their regulatory retention period expires.
4. **Sandboxing and Isolation**:
   * Process files in isolated environments (e.g., virtual machines or Docker containers) to reduce exposure.