Untitled1

September 17, 2024

1 Introduction

This Python script assumes PCI does not have a working report management system in place. This automates the extraction of key operational data from email files, such as Site Name, Nature of Work, and ISO Cause Code, commonly found in emails related to power generation events (e.g., maintenance, upgrades). The script decodes email content, extracts relevant data using regular expressions, and exports it to a CSV file for integration with Power BI and Tableau. It also visualizes the number of events per site, broken down by event type, through a stacked bar chart.

2 Business Case

The automated extraction and analysis of operational data provide possible business benefits including resource optimization, proactive maintenance: By identifying recurring issues, companies can prevent unplanned downtime and improve efficience, and data-driven decisions.

This solution enhances operational efficiency and supports long-term investment strategies by providing clear insights into site performance.

3 Importing Libraries

[1]:
```
import os
import zipfile
from email import policy
from email.parser import BytesParser
import collections
import re
import csv
import matplotlib.pyplot as plt
```

4 Extract Email Content

[2]:
```
# import dataset
zip_file_path="/Users/bandito/Documents/FA24/edf/PCI - September 2024.zip"
file_path = '/Users/bandito/Documents/FA24/edf/PCI - September 2024/'
```

```
# Function to extract email content from an .eml file with error handling for␣
↪encoding issues
def extract_email_content(file_path):
    with open(file_path, 'rb') as file:
        msg = BytesParser(policy=policy.default).parse(file)

    # Detect the charset from the headers, if present
    charset = msg.get_content_charset() if msg.get_content_charset() else␣
↪'utf-8'

    # Getting the body of the email (in case it's multipart)
    if msg.is_multipart():
        for part in msg.iter_parts():
            if part.get_content_type() == "text/plain":
                try:
                    return part.get_payload(decode=True).decode(charset)
                except UnicodeDecodeError:
                    return part.get_payload(decode=True).decode('ISO-8859-1',␣
↪errors='replace')  # Fallback to another encoding
    else:
        try:
            return msg.get_payload(decode=True).decode(charset)
        except UnicodeDecodeError:
            return msg.get_payload(decode=True).decode('ISO-8859-1',␣
↪errors='replace')
```

5 Extract Key Data Using Regex

[3]:
```
# Function to extract the necessary fields using regex
def extract_info_from_email(content):
    # Regex patterns to find site name, nature of work, and ISO cause code
    site_name_pattern = r"Please see outage request for\s(.+)"
    nature_of_work_pattern = r"Nature of Work:\s(.+)"
    iso_code_pattern = r"ISO Cause Code:\s(.+?)\n"

    site_name_match = re.search(site_name_pattern, content)
    nature_of_work_match = re.search(nature_of_work_pattern, content)
    iso_code_match = re.search(iso_code_pattern, content)

    site_name = site_name_match.group(1).strip() if site_name_match else "Pland␣
↪Derate"
    nature_of_work = nature_of_work_match.group(1).strip() if␣
↪nature_of_work_match else "Plant Derate"
    iso_code = iso_code_match.group(1).strip() if iso_code_match else "Plant␣
↪Derate"
```

```
    return site_name, nature_of_work, iso_code
```

6 Multiple Email Processing

[4]:
```
# Function to process all .eml files in a directory
def process_eml_files(directory):
    extracted_data = []
    for filename in os.listdir(directory):
        if filename.endswith(".eml"):
            file_path = os.path.join(directory, filename)
            content = extract_email_content(file_path)
            site_name, nature_of_work, iso_code =␣
 ↪extract_info_from_email(content)
            extracted_data.append((site_name, nature_of_work, iso_code))

    return extracted_data
```

7 Export .CSV

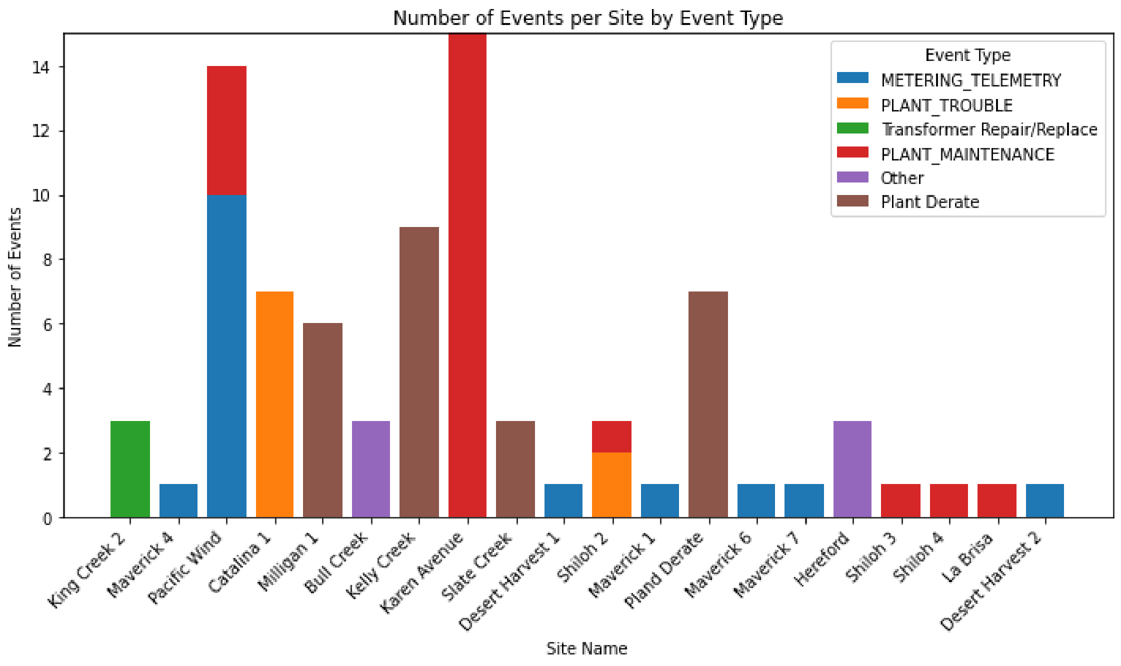
[5]:
```
# Function to export the extracted data to a CSV file
def export_to_csv(email_data, output_file):
    # Define the headers for the CSV
    headers = ['Site Name', 'Nature of Work', 'ISO Cause Code']

    # Write the data to a CSV file
    with open(output_file, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(headers)
        writer.writerows(email_data)

    print(f"Data successfully exported to {output_file}")
```

8 Create Charts

[6]:
```
# Visualization Function: Create a stacked bar chart for the event types per␣
 ↪site
def create_event_chart(email_data):
    # Organize data by site and event type
    site_event_counter = collections.defaultdict(lambda: collections.Counter())

    for site, event, iso_code in email_data:
        site_event_counter[site][event] += 1
    # Prepare data for plotting
    sites = list(site_event_counter.keys())
    event_types = list({event for events in site_event_counter.values() for␣
↪event in events})

    # Initialize a dictionary to hold counts for each event type per site
    event_data_per_site = {site: [site_event_counter[site].get(event, 0) for␣
↪event in event_types] for site in sites}

    # Create a stacked bar chart
    fig, ax = plt.subplots(figsize=(10, 6))

    bottom = [0] * len(sites)
    for i, event_type in enumerate(event_types):
        event_counts = [event_data_per_site[site][i] for site in sites]
        ax.bar(sites, event_counts, label=event_type, bottom=bottom)
        bottom = [x + y for x, y in zip(bottom, event_counts)]

    # Labeling the chart
    plt.xlabel('Site Name')
    plt.ylabel('Number of Events')
    plt.title('Number of Events per Site by Event Type')
    plt.xticks(rotation=45, ha="right")
    plt.legend(title='Event Type')

    # Display the plot
    plt.tight_layout()
    plt.show()
```

[7]:
```
# driver
directory_path = file_path  # Replace with your .eml files directory
email_data = process_eml_files(directory_path)

# Export the data to CSV for use in Power BI or Tableau
output_file = 'extracted_event_data.csv'
export_to_csv(email_data, output_file)

# Create a chart for event types per site
create_event_chart(email_data)
```

```
Data successfully exported to extracted_event_data.csv
```


Number of Events per Site by Event Type
Number of Events
Site Name
Event Type
METERING_TELEMETRY
PLANT_TROUBLE
Transformer Repair/Replace
PLANT_MAINTENANCE
Other
Plant Derate
King Creek 2
Maverick 4
Pacific Wind
Catalina 1
Milligan 1
Bull Creek
Kelly Creek
Karen Avenue
Slate Creek
Desert Harvest 1
Shiloh 2
Maverick 1
Pland Derate
Maverick 6
Maverick 7
Hereford
Shiloh 3
Shiloh 4
La Brisa
Desert Harvest 2