Reporte de Avance: Sistema de Monitoreo de Energía Solar

Minería de Datos

Integrantes:

Diego Carmona Bernal - 213498
Yumari Teresa Morales Mendoza - 211225
María José Domínguez Costa - 213457
Carlos Iván Cruz Zarmiento - 221236

Docente: Dr. Horacio Irán Solís Cisneros

Institución: Universidad Politécnica de Chiapas

Fecha de Presentación: 06 de marzo de 2025

Índice

1. Resumen

El presente documento describe el avance realizado en el desarrollo de un sistema de monitoreo para instalaciones de energía solar. El proyecto tiene como objetivo principal transformar los datos crudos de los inversores solares y sensores ambientales en un dashboard interactivo y funcional que permita monitorear el rendimiento de la instalación en tiempo real. Se detallan las fuentes de datos identificadas, los procesos de limpieza y procesamiento implementados, el diseño del front-end y la arquitectura propuesta para la solución completa. Adicionalmente, se presenta un cronograma de actividades para la siguiente fase del desarrollo, así como las conclusiones y observaciones preliminares.

2. Introducción

En el contexto actual, las instalaciones de energía solar requieren de sistemas eficientes para el monitoreo de su rendimiento y estado operativo. Con este objetivo, se ha iniciado el desarrollo de un sistema de monitoreo que permite transformar datos crudos en información valiosa para la toma de decisiones.

El sistema integra datos de diferentes fuentes para proporcionar una visión completa del funcionamiento de la instalación:

- Datos eléctricos de los inversores solares.
- Datos ambientales como temperatura, velocidad y dirección del viento.
- Datos de irradiancia solar en el plano de los paneles.

El presente reporte documenta el avance actual del proyecto, describiendo el trabajo realizado en la identificación y procesamiento de las fuentes de datos, el diseño del dashboard de visualización, y la planificación de las siguientes etapas de desarrollo.

3. Fuentes de Datos y Procesamiento

3.1. Identificación de Fuentes de Datos

Se han identificado tres fuentes principales de datos para el sistema:

1. **Datos eléctricos (2107_electrical_data.csv)**: Contiene mediciones de inversores solares, incluyendo corriente DC, voltaje DC, corriente AC, voltaje AC y potencia AC para 24 inversores.

2. **Datos ambientales (2107_environment_data.csv)**: Incluye mediciones de temperatura ambiente, velocidad del viento y dirección del viento.

3. **Datos de irradiancia (2107_irradiance_data.csv)**: Contiene mediciones de la irradiancia en el plano de los paneles (POA - Plane of Array).

3.2. Estructura de los Datasets

3.2.1. Datos eléctricos

- Formato: CSV con 120 columnas
- Frecuencia: Mediciones cada 5-15 minutos
- Columnas principales: measured_on, inv_XX_dc_current, inv_XX_dc_voltage, inv_XX_ac_current, inv_XX_ac_voltage, inv_XX_ac_power

3.2.2. Datos ambientales

- Formato: CSV con 4 columnas
- Frecuencia: Mediciones cada 15 minutos
- Columnas: measured_on, ambient_temperature_o_149575, wind_speed_o_149576, wind_direction_o_149577

3.2.3. Datos de irradiancia

- Formato: CSV con 2 columnas
- Frecuencia: Mediciones cada 5 minutos durante horas de luz
- Columnas: measured_on, poa_irradiance_o_149574

4. Procesamiento y Limpieza de Datos

Se ha desarrollado un módulo DataCleaner que implementa las siguientes funcionalidades para garantizar la calidad de los datos:

```
# Implementacion pendiente de ser agregada al codigo
class DataCleaner:
    def __init__(self, log_level=logging.INFO):
        # Configuracion del logger
        logging.basicConfig(level=log_level,
                            format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
        self.logger = logging.getLogger('DataCleaner')

    def remove_duplicates(self, df):
        """Elimina filas duplicadas del DataFrame."""
        initial_shape = df.shape
        df = df.drop_duplicates()
        removed = initial_shape[0] - df.shape[0]
        self.logger.info(f"Removed {removed} duplicate rows")
        return df

    def handle_missing_values(self, df, strategy='interpolate'):
        """
        Maneja valores faltantes segun la estrategia especificada.
        Estrategias disponibles: 'drop', 'interpolate', 'mean', 'median', 'zero'
        """
        missing_count = df.isna().sum().sum()
        self.logger.info(f"Found {missing_count} missing values")

        if strategy == 'drop':
            df = df.dropna()
        elif strategy == 'interpolate':
            df = df.interpolate(method='time')
        elif strategy == 'mean':
            df = df.fillna(df.mean())
        elif strategy == 'median':
```

```
32              df = df.fillna(df.median())
33          elif strategy == 'zero':
34              df = df.fillna(0)
35
36          remaining_missing = df.isna().sum().sum()
37          self.logger.info(f"Remaining missing values after {
    strategy}: {remaining_missing}")
38          return df
39
40      def detect_outliers(self, df, columns=None, method='iqr',
    threshold=1.5):
41          """
42          Detecta outliers en las columnas especificadas
    utilizando el metodo indicado.
43          Methods: 'iqr' (Interquartile Range), 'zscore'
44          """
45          if columns is None:
46              # Solo considerar columnas numericas
47              columns = df.select_dtypes(include=[np.number]).
    columns
48
49          outliers_summary = {}
50
51          for col in columns:
52              if method == 'iqr':
53                  Q1 = df[col].quantile(0.25)
54                  Q3 = df[col].quantile(0.75)
55                  IQR = Q3 - Q1
56                  lower_bound = Q1 - threshold * IQR
57                  upper_bound = Q3 + threshold * IQR
58                  outliers = df[(df[col] < lower_bound) | (df[col
    ] > upper_bound)][col]
59              elif method == 'zscore':
60                  from scipy import stats
61                  z_scores = np.abs(stats.zscore(df[col].dropna()
    ))
62                  outliers = df[col][z_scores > threshold]
63
64              outliers_summary[col] = len(outliers)
65              self.logger.info(f"Column {col}: {len(outliers)}
```

```
    outliers detected")
66
67          return outliers_summary
68
69      def standardize_timestamps(self, df, timestamp_col='
    measured_on', format=None):
70          """Estandariza la columna de timestamp al formato
    datetime."""
71          if timestamp_col in df.columns:
72              if format:
73                  df[timestamp_col] = pd.to_datetime(df[
    timestamp_col], format=format)
74              else:
75                  df[timestamp_col] = pd.to_datetime(df[
    timestamp_col])
76              self.logger.info(f"Standardized {timestamp_col} to
    datetime format")
77          return df
78
79      def resample_data(self, df, timestamp_col='measured_on',
    freq='15min', aggregation='mean'):
80          """
81          Remuestrea los datos a la frecuencia especificada.
82          Frecuencias comunes: '5min', '15min', '30min', '1H', '1
    D'
83          Agregaciones: 'mean', 'sum', 'max', 'min', 'median'
84          """
85          # Asegurarse de que el indice es un datetime
86          if df.index.dtype != 'datetime64[ns]':
87              if timestamp_col in df.columns:
88                  df = df.set_index(timestamp_col)
89              else:
90                  self.logger.error("DataFrame no tiene columna
    de timestamp ni indice datetime")
91                  return df
92
93          # Remuestrear
94          if aggregation == 'mean':
95              resampled = df.resample(freq).mean()
96          elif aggregation == 'sum':
```

```
97              resampled = df.resample(freq).sum()
98          elif aggregation == 'max':
99              resampled = df.resample(freq).max()
100         elif aggregation == 'min':
101             resampled = df.resample(freq).min()
102         elif aggregation == 'median':
103             resampled = df.resample(freq).median()
104
105         self.logger.info(f"Resampled data from {df.shape[0]} to
        {resampled.shape[0]} rows using {freq} frequency")
106         return resampled
```

Código 1: Implementación de la clase DataCleaner para limpieza de datos

4.1. Pasos de Limpieza Implementados

1. **Estandarización de timestamps**: Conversión de todas las marcas de tiempo al formato datetime de pandas.

2. **Manejo de valores faltantes**: Implementación de interpolación temporal para las series temporales.

3. **Eliminación de duplicados**: Detección y eliminación de registros duplicados.

4. **Detección de outliers**: Implementación de métodos IQR y Z-score para identificar valores atípicos.

5. **Remuestreo de datos**: Estandarización de la frecuencia de muestreo a 15 minutos para facilitar la integración de los tres datasets.

4.2. Métricas de Calidad de Datos

Cuadro 1: Resumen de la calidad de datos por dataset

Dataset	**Registros originales**	**Duplicados**	**Valores faltantes**	**Outliers**	**Registros finales**
Eléctrico	∼632,953	Pendiente	Pendiente	Pendiente	∼Pendiente
Ambiental	∼205,009	Pendiente	Pendiente	Pendiente	∼Pendiente
Irradiancia	∼531,020	Pendiente	Pendiente	Pendiente	∼Pendiente

5. Diseño e Implementación del Dashboard

5.1. Estructura del Front-end

Se ha desarrollado un prototipo funcional del front-end utilizando HTML, CSS y JavaScript. El dashboard incluye:

- **Cabecera**: Título y controles de fecha
- **Sistema de pestañas**: Dashboard principal, Estado de inversores, Alertas
- **Panel de métricas clave**: Potencia total AC, Irradiancia POA, Temperatura ambiente, Velocidad del viento
- **Visualizaciones principales**:
  - Gráfico de potencia AC por inversor
  - Gráfico de condiciones ambientales
  - Correlación entre irradiancia solar y producción de energía
- **Sección de alertas**: Visualización de alertas críticas, advertencias e informativas
- **Información detallada de inversores**: Estado operativo, parámetros eléctricos y eficiencia

5.2. Diseño Responsivo

El dashboard ha sido diseñado para adaptarse a diferentes tamaños de pantalla:

- Layout fluido con sistema de grid
- Menú colapsable para dispositivos móviles
- Visualizaciones que se ajustan automáticamente al ancho disponible

5.3. Visualizaciones Implementadas

Las visualizaciones han sido implementadas utilizando Chart.js y están preparadas para recibir datos dinámicos:

1. **Gráfico de líneas de potencia AC**: Muestra la evolución de la potencia generada por cada inversor a lo largo del tiempo.

2. **Gráfico de condiciones ambientales**: Combinación de temperatura, velocidad del viento y dirección del viento.

3. **Gráfico de correlación**: Muestra la relación entre la irradiancia solar y la producción de energía.

6. Arquitectura Propuesta de la Solución

La arquitectura del sistema se encuentra en fase de diseño y su implementación se realizará en etapas posteriores. Se tiene previsto desarrollar un backend en Python que exponga una API REST para el manejo de datos, la cual será consumida por un front-end construido con HTML, CSS y JavaScript. Además, se considera la posibilidad de integrar un proyecto en Laravel (PHP 8.1) para gestionar funcionalidades como el login, lo que permitiría una rápida implementación de la vista del dashboard y del sistema de alertas.

6.1. Diagrama de Arquitectura

Figura 1: Diagrama preliminar de la arquitectura propuesta (pendiente de implementación)

6.2. Componentes Principales

1. **Capa de Adquisición de Datos**:
   - Importación de archivos CSV
   - Programada para implementar conectores a APIs
2. **Capa de Procesamiento**:
   - Módulo DataCleaner para la limpieza y transformación
   - Sistema de normalización y almacenamiento en base de datos MySQL
3. **Capa de Análisis**:

- Cálculos de métricas de rendimiento (eficiencia, desviaciones)
- Algoritmos de detección de anomalías

4. **Capa de Presentación**:
   - Dashboard web interactivo
   - Sistema de alertas y notificaciones

6.3. Flujo de Datos

1. Los datos son capturados de las fuentes (archivos CSV)
2. El módulo DataCleaner procesa y limpia los datos
3. Los datos limpios se almacenan en una base de datos MySQL
4. Una API REST proporciona acceso a los datos procesados
5. El frontend consume la API y actualiza el dashboard

7. Planificación para la Siguiente Fase

7.1. Tareas Pendientes

1. **Integración del Backend**:
   - Desarrollar API REST con Python (Flask/FastAPI)
   - Implementar endpoints para cada tipo de visualización
2. **Migración a Base de Datos**:
   - Crear esquema de base de datos MySQL
   - Implementar scripts de carga y actualización
3. **Mejoras en Visualizaciones**:
   - Añadir interactividad a los gráficos
   - Implementar filtros y controles avanzados
4. **Sistema de Alertas**:

- Desarrollar lógica de detección de anomalías
- Implementar notificaciones en tiempo real

7.2. Cronograma Estimado

Cuadro 2: Cronograma de las actividades pendientes

Tarea	**Duración Estimada**	**Fecha Límite**
Implementación de la BD MySQL	5 días	15/03/2025
Desarrollo de API REST	7 días	22/03/2025
Integración Front-end/Back-end	5 días	27/03/2025
Sistema de alertas	3 días	30/03/2025
Pruebas finales	5 días	05/04/2025

8. Conclusiones y Observaciones

El avance actual representa aproximadamente el 40 % del desarrollo total del sistema. Se ha logrado:

1. Establecer la estructura de datos y los procesos de limpieza
2. Desarrollar el prototipo de interfaz de usuario
3. Definir la arquitectura completa de la solución

Los retos identificados para las siguientes fases incluyen:

- La integración eficiente entre las diferentes fuentes de datos
- La optimización del rendimiento para manejar grandes volúmenes de datos
- La implementación de algoritmos de detección de anomalías precisos

El proyecto avanza según lo planificado y se prevé cumplir con todos los requisitos dentro del plazo establecido.

9. Anexos

9.1. Capturas de Pantalla del Dashboard

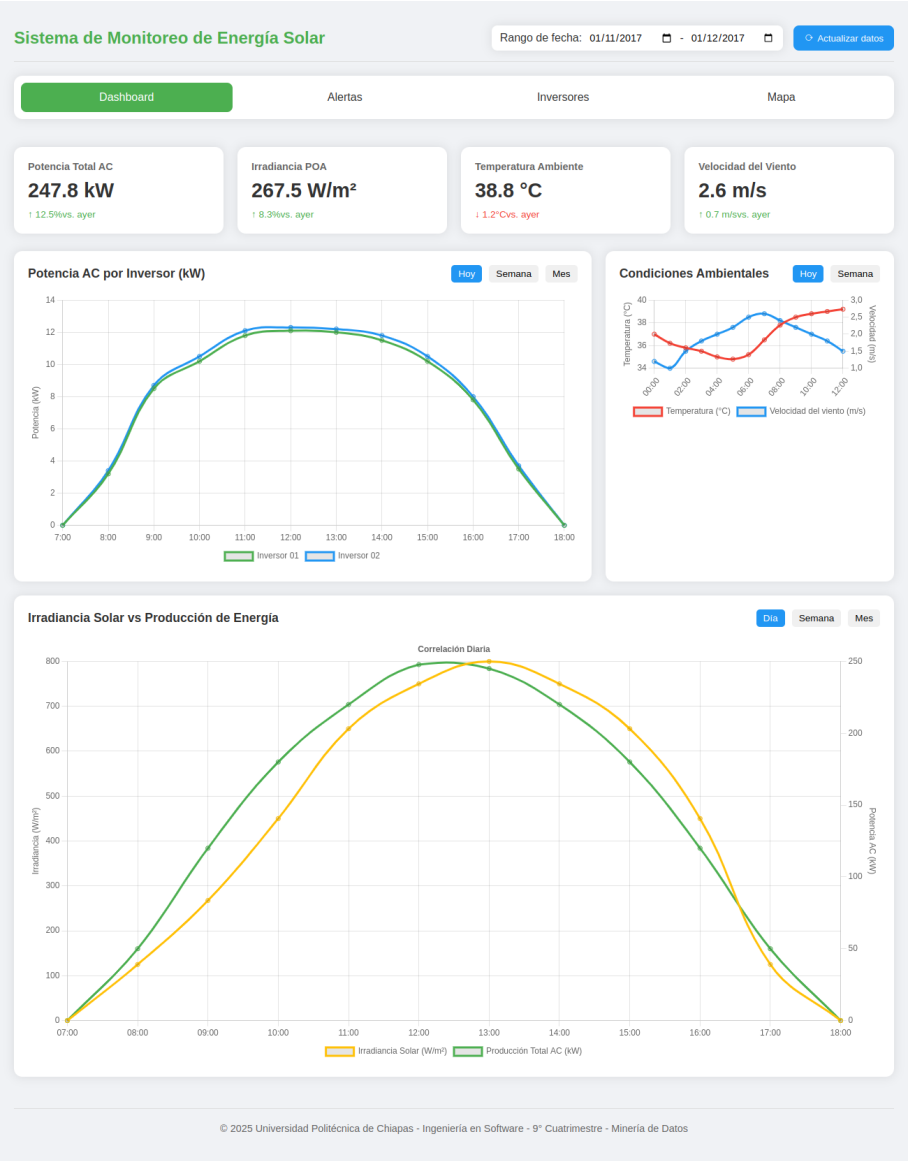

Figura 2: Vista del panel principal

Figura 3: Vista de métricas

Figura 4: Vista de alertas

9.2. Ejemplos de Código

```
1 // Configuracion del gr fico de potencia AC
2 const ctx = document.getElementById('powerChart').getContext('2
    d');
3 const powerChart = new Chart(ctx, {
4     type: 'line',
5     data: {
6         labels: timeLabels,
7         datasets: inverters.map(inv => ({
8             label: `Inversor ${inv}`,
9             data: powerData[inv],
10            borderColor: getColor(inv),
11            tension: 0.1,
```

```
12              fill: false
13          }))
14      },
15      options: {
16          responsive: true,
17          maintainAspectRatio: false,
18          interaction: {
19              mode: 'index',
20              intersect: false,
21          },
22          scales: {
23              y: {
24                  title: {
25                      display: true,
26                      text: 'Potencia AC (kW)'
27                  }
28              },
29              x: {
30                  title: {
31                      display: true,
32                      text: 'Hora'
33                  }
34              }
35          }
36      }
37 });
```

Código 2: Ejemplo de codigo para la visualizacion de potencia AC

9.3. Muestras de Datos Procesados: Condiciones Ambientales y Gráfico de Potencia AC

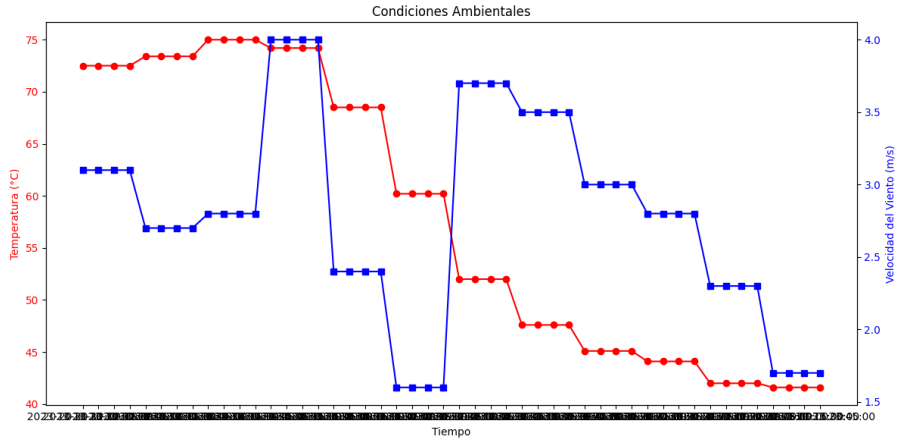

Figura 5: Gráfico de las condiciones ambientales (temperatura, viento)

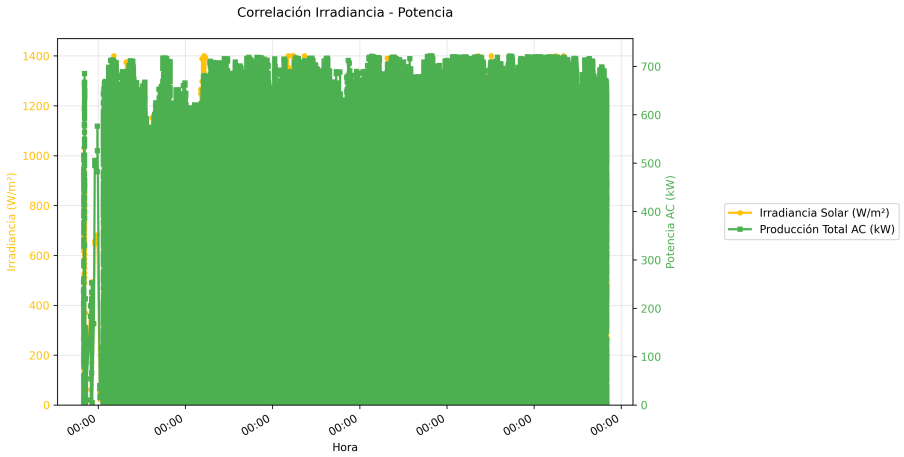

Figura 6: Correlación entre irradiancia solar y potencia generada

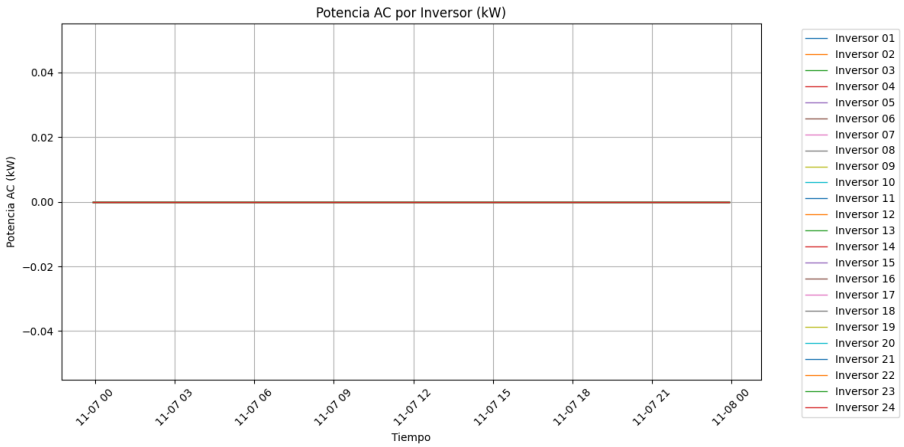

Figura 7: Gráfico de potencia AC por inversor