Comprehensive Hands-On Guide: Data Cleaning and Preprocessing for IoT Well Log Sensor Data

1 Objective

This guide covers the steps required to clean and preprocess well log data collected via IoT sensors. We will handle common data issues such as missing values, outliers, noise, and normalization. After this process, the data will be ready for further analysis or machine learning models.

2 Setup and Required Libraries

Before starting, install the necessary Python libraries:

```
pip install numpy pandas matplotlib seaborn scipy scikit-learn statsmodels
```

3 1. Loading and Inspecting the Well Log Data

Load the well log dataset from a CSV file containing columns like Depth, Gamma Ray, Resistivity, Sonic, and Temperature logs.

```
import pandas as pd

# Load the dataset (replace 'well_log_iot.csv' with your actual file path)
data = pd.read_csv('well_log_iot.csv')

# Display the first few rows of the dataset
print(data.head())

# Display basic statistics
print(data.describe())
```

4 2. Handling Missing Data

Missing data can arise due to IoT sensor failures or communication interruptions. Let's first identify and handle missing data.

4.1 2.1. Identifying Missing Data

We can check for missing values in the dataset and visualize them using a heatmap.

```
# Checking for missing values
missing_values = data.isnull().sum()
print("Missing values per column:\n", missing_values)

# Visualizing missing data
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
sns.heatmap(data.isnull(), cbar=False, cmap='magma')
plt.title("Heatmap of Missing Values")
plt.show()
```

4.2 2.2. Handling Missing Data

We can handle missing data using several methods such as dropping rows, filling with median, or interpolating.

```
# Option 1: Dropping rows with too many missing values
data_cleaned = data.dropna(thresh=data.shape[1] - 1)  # Keep rows with at least N-1 non-null values

# Option 2: Fill missing values with the median of each column
data_cleaned = data.fillna(data.median())

# Option 3: Interpolating missing values (useful for time-series or depth-indexed data)
data_cleaned = data.interpolate()

# Option 4: KNN Imputation for more advanced imputation
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=3)
data_cleaned = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

# Check the cleaned dataset
print(data_cleaned.head())
```

5 3. Detecting and Treating Outliers

Outliers in IoT well log data can be due to sensor malfunctions or geological anomalies. We will detect them using visualizations and statistical methods, and then treat them.

5.1 3.1. Visualizing Outliers with Boxplots

```
# Visualizing outliers using boxplots
plt.figure(figsize=(12,6))
sns.boxplot(data=data[['Gamma Ray', 'Resistivity', 'Sonic',
    'Temperature']])
plt.title('Boxplot for Outlier Detection')
plt.show()
```

5.2 3.2. Detecting Outliers Using Z-Score and IQR

We can detect outliers statistically using Z-scores and the interquartile range (IQR) method.

```
# Z-Score method to detect outliers
from scipy import stats
import numpy as np

z_scores = np.abs(stats.zscore(data[['Gamma Ray', '
    Resistivity', 'Sonic']]))
outliers_z = np.where(z_scores > 3)
print("Outliers detected using Z-score at rows:", outliers_z
    )

# IQR Method to detect outliers
Q1 = data[['Gamma Ray', 'Resistivity', 'Sonic']].quantile
    (0.25)
Q3 = data[['Gamma Ray', 'Resistivity', 'Sonic']].quantile
    (0.75)
IQR = Q3 - Q1

outliers_iqr = (data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5
     * IQR))
print("Outliers detected using IQR method:\n", outliers_iqr.
    sum())
```

5.3 3.3. Treating Outliers

You can remove outliers or cap them to a certain range.

```
# Option 1: Removing outliers (using Z-Score or IQR
    detection)
```

```
data_cleaned = data[(z_scores < 3).all(axis=1)]

# Option 2: Capping outliers to the 1st and 99th percentile
def cap_outliers(df, column):
    lower = df[column].quantile(0.01)
    upper = df[column].quantile(0.99)
    df[column] = np.clip(df[column], lower, upper)
    return df

for col in ['Gamma Ray', 'Resistivity', 'Sonic']:
    data_cleaned = cap_outliers(data_cleaned, col)

print(data_cleaned.head())
```

6 4. Noise Reduction

Sensor data from IoT systems often contains noise, which can be reduced using smoothing techniques like moving averages or filters.

6.1 4.1. Moving Average Smoothing

```
# Apply moving average smoothing with a window of 5
data_cleaned['Gamma Ray (Smoothed)'] = data_cleaned['Gamma Ray'].rolling(window=5).mean()
data_cleaned['Resistivity (Smoothed)'] = data_cleaned['Resistivity'].rolling(window=5).mean()

# Plot the original vs smoothed data
plt.figure(figsize=(12,6))
plt.plot(data_cleaned['Depth'], data_cleaned['Gamma Ray'], label='Original Gamma Ray', alpha=0.5)
plt.plot(data_cleaned['Depth'], data_cleaned['Gamma Ray (Smoothed)'], label='Smoothed Gamma Ray', color='red')
plt.xlabel('Depth')
plt.ylabel('Gamma Ray')
plt.legend()
plt.title('Gamma Ray Before and After Smoothing')
plt.show()
```

6.2 4.2. Low-Pass Filter

A low-pass filter removes high-frequency noise while preserving low-frequency signals.

```
from scipy.signal import butter, filtfilt

```

```
# Define a Butterworth low-pass filter
def butter_lowpass(cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=
        False)
    return b, a

def lowpass_filter(data, cutoff, fs, order=5):
    b, a = butter_lowpass(cutoff, fs, order=order)
    y = filtfilt(b, a, data)
    return y

# Apply the low-pass filter
filtered_resistivity = lowpass_filter(data_cleaned['
    Resistivity'], cutoff=0.1, fs=1.0)

# Plot original vs filtered resistivity
plt.figure(figsize=(12,6))
plt.plot(data_cleaned['Depth'], data_cleaned['Resistivity'],
     label='Original Resistivity', alpha=0.5)
plt.plot(data_cleaned['Depth'], filtered_resistivity, label=
    'Filtered Resistivity', color='red')
plt.xlabel('Depth')
plt.ylabel('Resistivity')
plt.legend()
plt.title('Resistivity Before and After Low-Pass Filtering')
plt.show()
```

7 5. Data Normalization and Standardization

To ensure the data is on the same scale, we normalize or standardize it.

7.1 5.1. Normalization (Min-Max Scaling)

```
from sklearn.preprocessing import MinMaxScaler

# Normalize to the range [0, 1]
scaler = MinMaxScaler()
data_normalized = pd.DataFrame(scaler.fit_transform(
    data_cleaned[['Gamma Ray', 'Resistivity', 'Sonic']]),
    columns=['Gamma Ray', 'Resistivity', 'Sonic'])

print(data_normalized.head())
```

7.2 5.2. Standardization

```
from sklearn.preprocessing import StandardScaler

# Standardize to have a mean of 0 and standard deviation of 1
scaler = StandardScaler()
data_standardized = pd.DataFrame(scaler.fit_transform(data_cleaned[['Gamma Ray', 'Resistivity', 'Sonic']]), columns=['Gamma Ray', 'Resistivity', 'Sonic'])

print(data_standardized.head())
```

8 6. Feature Engineering

We can create new features based on existing logs, such as estimating lithology from gamma ray readings or calculating porosity from sonic logs.

8.1 6.1. Deriving Lithology from Gamma Ray

Lithology is often inferred from gamma ray values, where high values indicate shale and low values indicate sand.

```
# Creating a simple lithology feature based on Gamma Ray threshold
data_cleaned['Lithology'] = data_cleaned['Gamma Ray'].apply(lambda x: 'Shale' if x > 75 else 'Sand')

print(data_cleaned[['Depth', 'Gamma Ray', 'Lithology']].head())
```

8.2 6.2. Estimating Porosity from Sonic Log

You can estimate porosity using sonic log data with empirical formulas.

```
# Simple porosity calculation from the sonic log
data_cleaned['Porosity'] = (data_cleaned['Sonic'] - 50) / 150  # Simplified formula

print(data_cleaned[['Depth', 'Sonic', 'Porosity']].head())
```

9 7. Visualization of Cleaned and Preprocessed Data

After cleaning and preprocessing the data, you can visualize it.

```
# Plot final cleaned data
plt.figure(figsize=(12,8))

plt.subplot(3, 1, 1)
plt.plot(data_cleaned['Depth'], data_cleaned['Gamma Ray'],
    label='Gamma Ray', color='blue')
plt.xlabel('Depth')
plt.ylabel('Gamma Ray')
plt.title('Gamma Ray vs Depth')

plt.subplot(3, 1, 2)
plt.plot(data_cleaned['Depth'], data_cleaned['Resistivity'],
     label='Resistivity', color='green')
plt.xlabel('Depth')
plt.ylabel('Resistivity')
plt.title('Resistivity vs Depth')

plt.subplot(3, 1, 3)
plt.plot(data_cleaned['Depth'], data_cleaned['Sonic'], label
    ='Sonic', color='red')
plt.xlabel('Depth')
plt.ylabel('Sonic')
plt.title('Sonic vs Depth')

plt.tight_layout()
plt.show()
```

10 Conclusion

In this comprehensive hands-on session, we:

- Loaded and inspected IoT well log data.
- Handled missing values using imputation and interpolation.
- Detected and treated outliers using statistical methods.
- Reduced noise with smoothing and low-pass filtering.
- Normalized and standardized the data.
- Engineered new features such as lithology and porosity.
- Visualized the final cleaned data.

These steps prepare IoT well log data for machine learning models or further geophysical analysis.