DATA SCIENCE INTERNSHIP

TASK1 Data Preprocessing

Handle missing values and outlines appropriately.Normalize or scale features as needed . split the data into training and testing sets .

python

Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

Load your dataset

Replace 'your_dataset.csv' with the path to your dataset file

df = pd.read_csv('your_dataset.csv')

1. Handle Missing Values

Let's impute missing values with the mean of the column (you can also use median or mode)

imputer = SimpleImputer(strategy='mean')

df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

2. Normalize/Scale the Features (if necessary)

We will use StandardScaler to scale numerical features

scaler = StandardScaler()

Assuming all columns are numeric, if not select numeric columns using df.select_dtypes(include=[float, int])

scaled_features = scaler.fit_transform(df_imputed)

3. Split the Data into Training and Testing Sets

Let's assume that the target variable (the variable you want to predict) is 'target_column'

X = df_imputed.drop('target_column', axis=1) # Features

y = df_imputed['target_column'] # Target variable

Splitting data into training (80%) and testing (20%) sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Optionally, you can use the scaled data:

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

Now, X_train_scaled and X_test_scaled are ready to be used for model training

Task2

Feature Engineering

Create additional features that might be useful for prediction equipment failure . Consider time based features rolling statistics and any other relevant transformation.

``python

Import necessary libraries

import pandas as pd

import numpy as np

Sample DataFrame (replace this with your actual dataset)

The dataset should have a 'timestamp' column and columns for equipment data (e.g., temperature, pressure)

df = pd.read_csv('your_equipment_data.csv')

Convert timestamp column to datetime format if not already

df['timestamp'] = pd.to_datetime(df['timestamp'])

1. Time-Based Features

Create a feature for the hour of the day, day of the week, and month

df['hour'] = df['timestamp'].dt.hour

df['day_of_week'] = df['timestamp'].dt.dayofweek # Monday=0, Sunday=6

df['month'] = df['timestamp'].dt.month

Calculate the time difference from the last maintenance (example)

df['time_since_last_maintenance'] = (df['timestamp'] - df['timestamp'].shift(1)).dt.total_seconds()

2. Rolling Statistics

Adding rolling statistics for key variables (e.g., temperature, pressure)

This can help capture patterns or trends over time.

For example, create a rolling window of 3 periods (you can adjust the window size)

df['temperature_rolling_mean'] = df['temperature'].rolling(window=3).mean()

df['pressure_rolling_mean'] = df['pressure'].rolling(window=3).mean()

Create rolling standard deviation (for variability) over a 3-period window

df['temperature_rolling_std'] = df['temperature'].rolling(window=3).std()

df['pressure_rolling_std'] = df['pressure'].rolling(window=3).std()

Calculate rolling sum to capture long-term trends

df['temperature_rolling_sum'] = df['temperature'].rolling(window=3).sum()

3. Cumulative Features

Cumulative sum of features like temperature or pressure, which could be useful to detect the effect of prolonged exposure

df['temperature_cumulative_sum'] = df['temperature'].cumsum()

4. Lag Features

Lagging features help in capturing temporal dependencies.

For example, lagging the temperature and pressure by 1 period (or more)

df['temperature_lag_1'] = df['temperature'].shif

t(1)

df['pressure_lag_1'] = df['pressure'].shift(1)

Lagging can also be done for other features like 'time_since_last_failure' if present in the dataset

df['time_since_last_failure'] = df['failure_timestamp'].shift(1)

5. Failure History Features (if applicable)

If you have the target column that indicates failure ('failure_flag'), you can create features based on previous failures.

df['failure_flag_lag_1'] = df['failure_flag'].shift(1)

df['failure_flag_lag_2'] = df['failure_flag'].shift(2)

6. Interaction Features

Interaction features can capture relationships between multiple variables. For instance, combining temperature and pressure.

df['temperature_pressure_interaction'] = df['temperature'] * df['pressure']

7. Other transformations: Exponentiation, logarithm, etc.

For instance, log-transform the temperature data if it's exponentially distributed.

df['log_temperature'] = np.log(df['temperature'] + 1) # Adding 1 to avoid log(0)

8. Target Encoding (if applicable)

If you have categorical features like 'equipment_type', you can encode it based on the target variable.

For example, encoding 'equipment_type' with the mean failure rate for each type of equipment.

df['equipment_type_failure_rate'] = df.groupby('equipment_type')['failure_flag'].transform('mean')