```
import warnings ; warnings.filterwarnings(’ignore’)

# Data manipulation
import numpy as np
import pandas as pd ; pd.set_option(’display.max_columns’, None) ; pd.set_option(’display.max_rows’, 4)

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns ; color_pal = sns.color_palette("husl", 9) ; plt.style.use(’fivethirtyeight’)

# Utilities
from datetime import datetime, date
import math
import os
import re
import missingno as msno
consumptions = pd.read_csv(’../data/raw/energy/household_power_consumption.zip’, sep =’;’, header=0,
na_values=’?’,
                          dtype={’Date’:str, ’Time’:str, ’Global_active_power’:np.float64},
                          infer_datetime_format=False)

# Standardise column names using lower case
consumptions.rename(
    columns = {
        ’Date’:’date’,
        ’Time’:’time’,
        ’Global_active_power’:’total_consumption’
        },
    inplace=True
)

# Define the dataframe index based on the timestamp (date-time)
consumptions.index = pd.to_datetime(
    consumptions.date + "-" + consumptions.time,
    format = "%d/%m/%Y-%H:%M:%S"
)

# Drop the date and time variables that are now redondant with the index
consumptions.drop(columns=[’date’, ’time’], inplace=True)

# We resample for you to continue the exercise
consumptions_df = consumptions.resample(’D’).sum()
consumptions_df.tail(3)
consumptions_df.head(2)
consumptions_df.tail(2)
consumptions_df.columns
print(consumptions_df.shape)
consumptions_df.info()
plt.figure(figsize=(20,5))
plt.title(’Electric Power Consumption Over Time’)
plt.xlabel(’Date’)
plt.ylabel(’Total Consumption (kWh)’)
plt.plot(consumptions_df[’total_consumption’])
plt.show()
msno.matrix(consumptions_df)
```

```
# Function to calculate the rho association metric
def calculate_rho(grouped_data, overall_mean):
    sum_of_squares_within = sum(grouped_data.apply(lambda x: len(x) * (x.mean() - overall_mean)**2))
    total_sum_of_squares = sum((consumptions_df_copy[’total_consumption’] - overall_mean)**2)
    rho = sum_of_squares_within / total_sum_of_squares
    return rho

# Copy the data for transformations
consumptions_df_copy = consumptions_df.copy()
consumptions_df_copy[’dayofweek’] = consumptions_df_copy.index.dayofweek
consumptions_df_copy[’month’] = consumptions_df_copy.index.month
consumptions_df_copy[’quarter’] = consumptions_df_copy.index.quarter
consumptions_df_copy[’year’] = consumptions_df_copy.index.year

# Overall mean of total consumption
overall_mean = consumptions_df_copy[’total_consumption’].mean()

# Create a figure with multiple subplots
fig, axes = plt.subplots(2, 2, figsize=(20, 8))

# List of categories
categories = [’dayofweek’, ’month’, ’quarter’, ’year’]
category_labels = [
    [’Mon’, ’Tue’, ’Wed’, ’Thu’, ’Fri’, ’Sat’, ’Sun’],
    [’Jan’, ’Feb’, ’Mar’, ’Apr’, ’May’, ’Jun’, ’Jul’, ’Aug’, ’Sep’, ’Oct’, ’Nov’, ’Dec’],
    [’Q1’, ’Q2’, ’Q3’, ’Q4’],
    range(consumptions_df_copy[’year’].min(), consumptions_df_copy[’year’].max() + 1)
]

# Plot for each category
for i, (category, labels) in enumerate(zip(categories, category_labels)):
    ax = axes[i // 2, i % 2]
    sns.boxplot(data=consumptions_df_copy, x=category, y=’total_consumption’, ax=ax,
palette=color_pal)
    ax.grid(True, linestyle=’--’, alpha=0.7)

    # Calculate the rho value for the category
    grouped = consumptions_df_copy.groupby(category)[’total_consumption’]
    rho = calculate_rho(grouped, overall_mean)

    # Add the rho value as text on the plot
    ax.text(0.95, 0.95, f’ρ = {rho:.2f}',
            transform=ax.transAxes,
            horizontalalignment='right',
            verticalalignment='top',
            fontsize=12,
            bbox=dict(facecolor='white', alpha=0.5))

    # Add a red line for the overall mean
    ax.axhline(overall_mean, color='red', linestyle='--')

    ax.set_title(f'Electric Power Consumption by {category.capitalize()}')
    ax.set_xlabel(category.capitalize())
    ax.set_ylabel('Total Consumption (kWh)')
    ax.set_xticklabels(labels)
    ax.tick_params(axis='both', which='major', labelsize=10)
plt.tight_layout()
plt.show()
correlation_matrix = consumptions_df.corr()
plt.figure(figsize=(6, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix of Features")
plt.show()
threshold = 0.5
target_correlations = correlation_matrix['total_consumption'].drop('total_consumption', axis=0).abs()
highly_correlated_features = target_correlations[target_correlations > threshold]
print("Most correlated features with 'total_consumption':")
print(highly_correlated_features.sort_values(ascending=False))
french_holidays_df = pd.read_csv("../data/raw/holidays/jours_feries_metropole.csv",
                                 parse_dates=['date'])
french_holidays_df.head(3)
start_date = french_holidays_df.date.min()
end_date = french_holidays_df.date.max()
print(f'Data spans from {str(start_date)[:10]} to {str(end_date)[:10]}')
weather_dictionary = {}
data_directory = "../data/raw/weather/"
for file_name in os.listdir(data_directory):
    if file_name.endswith(".csv"):
        weather_dictionary[file_name] = pd.read_csv(os.path.join(data_directory, file_name),
                                                    parse_dates=['datetime', 'sunrise', 'sunset'],
                                                    index_col='datetime')
print(weather_dictionary.keys())
weather_df = pd.concat([weather_df for df_name, weather_df in weather_dictionary.items()], axis=0)
weather_df.tail(2)
msno.matrix(weather_df)
plt.figure(figsize=(5, 3))
plt.show()
def clean_string(s):
    """
    Cleans a string: replaces spaces with underscores, removes special characters, and converts to
lowercase.
    """
    return re.sub(r'[^a-zA-Z0-9\s]', '', s.replace(' ', '_')).lower()
def calculate_day_length(df, sunrise_col='sunrise', sunset_col='sunset'):
    """
    Adds 'day_length' to df calculated from 'sunrise' and 'sunset', and drops these columns.
    """
    df[sunrise_col] = pd.to_datetime(df[sunrise_col], format='%H:%M:%S').dt.time
    df[sunset_col] = pd.to_datetime(df[sunset_col], format='%H:%M:%S').dt.time
    df['day_length'] = ((pd.to_datetime(df[sunset_col].astype(str)) -
pd.to_datetime(df[sunrise_col].astype(str))).dt.total_seconds()) / 3600.0
    return df.drop([sunrise_col, sunset_col], axis=1)

def preprocess_weather_data(df, start_date, end_date, columns_to_keep, column_to_encode):
    """
    Preprocesses weather data: sorts by index, filters by date, selects columns, encodes a column, and
calculates day length.
    """
    df.sort_index(inplace=True)
    df_selected = df[columns_to_keep].copy()
    df_filtered = df_selected[(df_selected.index >= start_date) & (df_selected.index <= end_date)].copy()
    df_filtered[column_to_encode] = df_filtered[column_to_encode].apply(clean_string)
    dummies = pd.get_dummies(df_filtered[column_to_encode], prefix=column_to_encode)
    df_encoded = pd.concat([df_filtered, dummies], axis=1).drop(column_to_encode, axis=1)
    return calculate_day_length(df_encoded)
columns_to_keep = ['tempmax', 'tempmin', 'temp', 'feelslikemax', 'feelslikemin', 'feelslike',
                   'dew', 'humidity', 'precip', 'precipprob', 'precipcover', 'snow', 'snowdepth',
                   'windgust', 'windspeed', 'winddir', 'sealevelpressure', 'cloudcover',
                   'visibility', 'sunrise', 'sunset', 'moonphase', 'conditions']
start_date='2006-12-16'
end_date='2010-11-26'
column_to_encode='conditions'
processed_weather_df = preprocess_weather_data(df=weather_df,
                                               start_date=start_date,
                                               end_date=end_date,
                                               columns_to_keep=columns_to_keep,
                                               column_to_encode=column_to_encode)
processed_weather_df.head(2)
processed_weather_df.head(1)
consumptions_df.head(1)
weather_and_consumption_df = pd.merge(consumptions_df, processed_weather_df, left_index=True,
right_index=True)
weather_and_consumption_df.head(1)
french_holidays_set = set(french_holidays_df.date)
weather_and_consumption_df['is_holiday'] =
weather_and_consumption_df.index.isin(french_holidays_set)
weather_and_consumption_df.head(2)
weather_and_consumption_df[weather_and_consumption_df.index=='2007-05-01'].is_holiday
correlation_matrix = weather_and_consumption_df.corr()
threshold = 0.5
highly_correlated_features = correlation_matrix['total_consumption'].drop('total_consumption').abs()
highly_correlated_features = highly_correlated_features[highly_correlated_features >
threshold].sort_values(ascending=False)
print("Most correlated features with 'total_consumption':")
for feature in highly_correlated_features.index:
    print(" " + feature)
weather_and_consumption_df.head(1)
weather_and_consumption_df.to_csv('../data/processed/weather_and_consumption.csv', index=True)
```