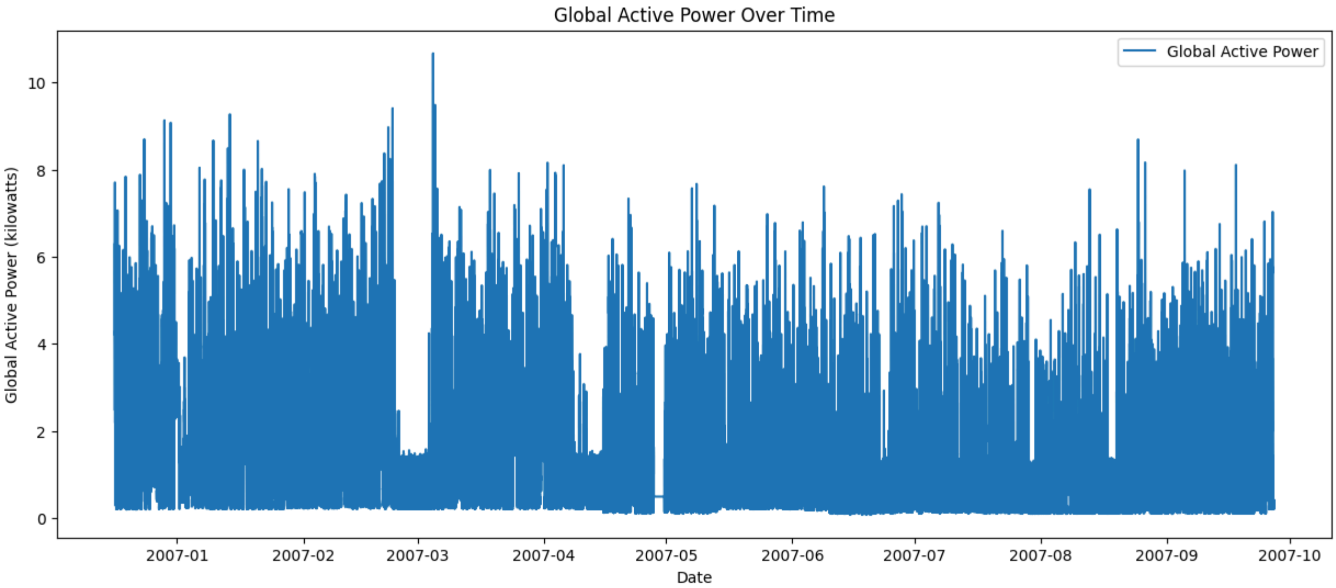
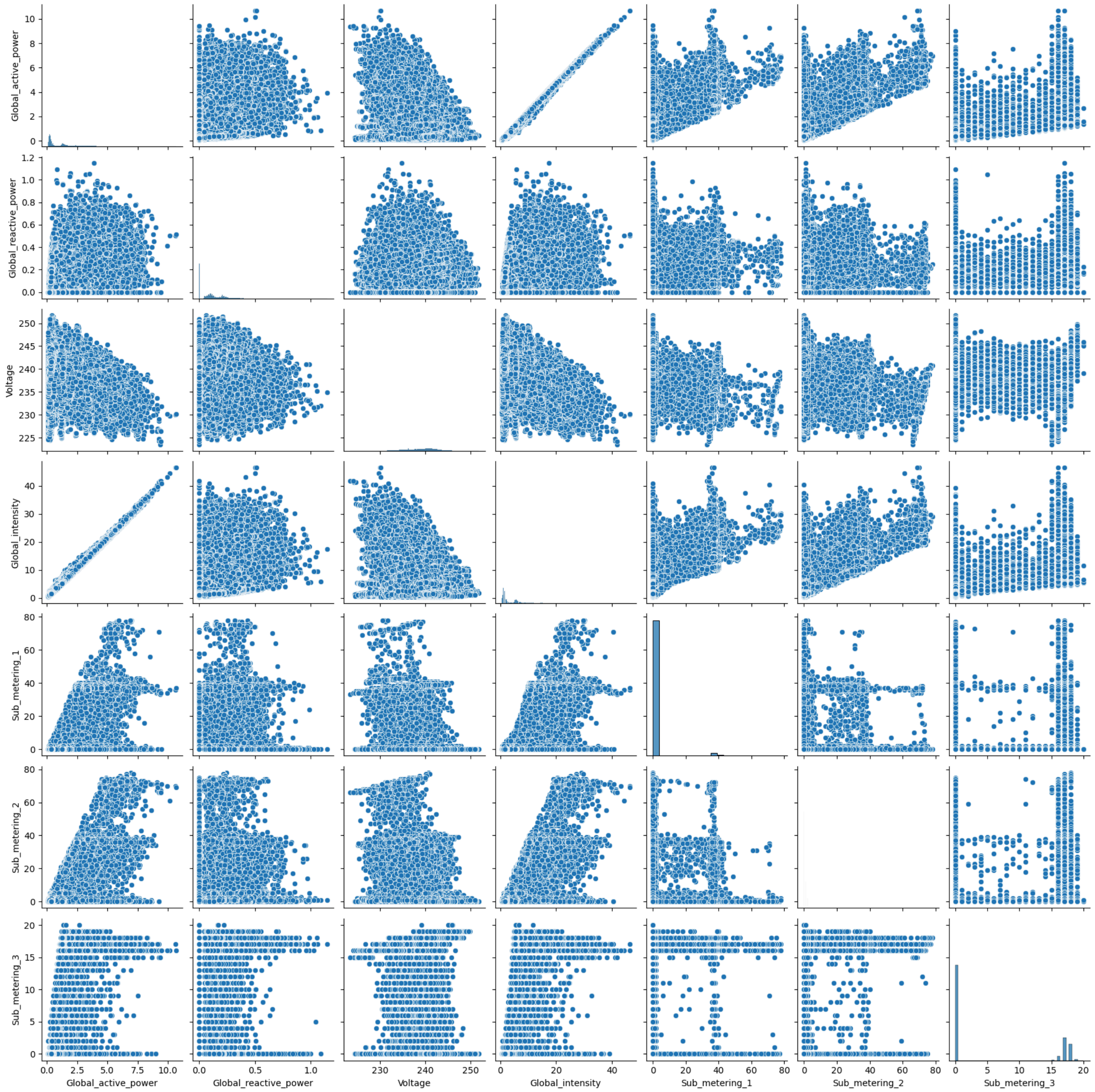
```
# Data preprocessing
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/household_power_consumption.txt'
df = pd.read_csv(file_path, sep=';', low_memory=False)

# Handle missing values (forward fill)
df.fillna(method='ffill', inplace=True)

# Combine Date and Time into a single datetime column
df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], format='%d/%m/%Y %H:%M:%S', errors='coerce')
df.set_index('Datetime', inplace=True)

# Drop the original Date and Time columns
df.drop(['Date', 'Time'], axis=1, inplace=True)
# Convert all columns to numeric, coercing errors to NaN
df = df.apply(pd.to_numeric, errors='coerce')

# Ensure numeric columns are used for quantile calculations
numeric_cols = df.select_dtypes(include=[np.number]).columns

# Detect outliers using IQR method
Q1 = df[numeric_cols].quantile(0.25)
Q3 = df[numeric_cols].quantile(0.75)
IQR = Q3 - Q1

# Define a mask for filtering out the outliers
outlier_mask = (df[numeric_cols] < (Q1 - 1.5 * IQR)) | (df[numeric_cols] > (Q3 + 1.5 * IQR))

# Replace outliers with NaN, then forward fill
df[numeric_cols][outlier_mask] = pd.NA
df.fillna(method='ffill', inplace=True)

# Display the first few rows of the updated dataframe
print(df.head())
```

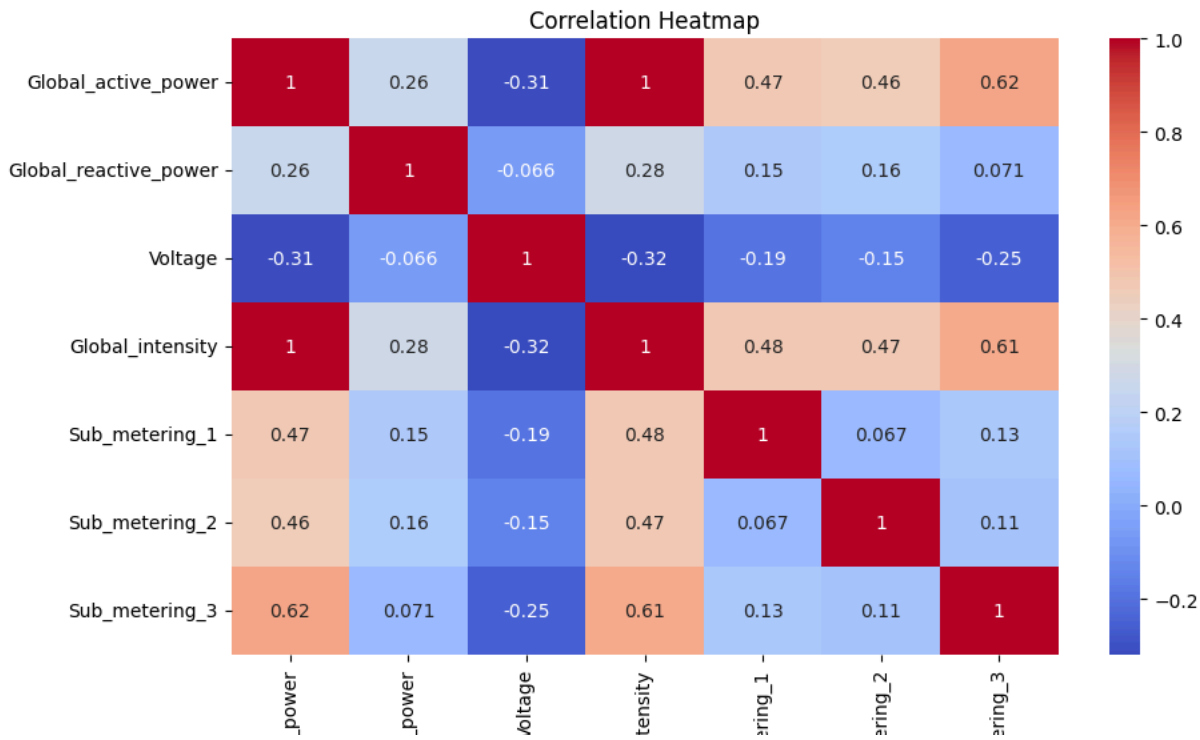
```
                     Global_active_power  Global_reactive_power  Voltage  \
Datetime
2006-12-16 17:24:00                4.216                  0.418   234.84
2006-12-16 17:25:00                5.360                  0.436   233.63
2006-12-16 17:26:00                5.374                  0.498   233.29
2006-12-16 17:27:00                5.388                  0.502   233.74
2006-12-16 17:28:00                3.666                  0.528   235.68

                     Global_intensity  Sub_metering_1  Sub_metering_2  \
Datetime
2006-12-16 17:24:00              18.4             0.0             1.0
2006-12-16 17:25:00              23.0             0.0             1.0
2006-12-16 17:26:00              23.0             0.0             2.0
2006-12-16 17:27:00              23.0             0.0             1.0
2006-12-16 17:28:00              15.8             0.0             1.0

                     Sub_metering_3
Datetime
2006-12-16 17:24:00            17.0
2006-12-16 17:25:00            16.0
2006-12-16 17:26:00            17.0
2006-12-16 17:27:00            17.0
2006-12-16 17:28:00            17.0
```

```
#step2 EDA
# Uncover Patterns, Trends, and Seasonality
import matplotlib.pyplot as plt

# Plot Global_active_power over time
plt.figure(figsize=(15, 6))
plt.plot(df.index, df['Global_active_power'], label='Global Active Power')
plt.xlabel('Date')
plt.ylabel('Global Active Power (kilowatts)')
plt.title('Global Active Power Over Time')
plt.legend()
plt.show()
```




```
#visualize relationships between features
import seaborn as sns

# Pair plot
sns.pairplot(df[['Global_active_power', 'Global_reactive_power', 'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2', 'Sul
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```


Global_active_power
Global_reactive_power
Voltage
Global_intensity
Sub_metering_1
Sub_metering_2
Sub_metering_3


Correlation Heatmap
Global_active_power
Global_reactive_power
Voltage
Global_intensity
Sub_metering_1
Sub_metering_2
Sub_metering_3
1 0.26 -0.31 1 0.47 0.46 0.62
0.26 1 -0.066 0.28 0.15 0.16 0.071
-0.31 -0.066 1 -0.32 -0.19 -0.15 -0.25
1 0.28 -0.32 1 0.48 0.47 0.61
0.47 0.15 -0.19 0.48 1 0.067 0.13
0.46 0.16 -0.15 0.47 0.067 1 0.11
0.62 0.071 -0.25 0.61 0.13 0.11 1
_power
_power
Voltage
tensity
ering_1
ering_2
ering_3
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


```
# Time series forecasting with ARIMA model step 3
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Ensure 'Global_active_power' is converted to numeric and drop NaN values
df['Global_active_power'] = pd.to_numeric(df['Global_active_power'], errors='coerce')
df = df.dropna(subset=['Global_active_power'])

# Define the model
arima_model = ARIMA(df['Global_active_power'], order=(5, 1, 0))

# Fit the model
arima_result = arima_model.fit()

# Forecast
forecast = arima_result.forecast(steps=30)
conf_int = arima_result.get_forecast(steps=30).conf_int()

# Plot the forecast
plt.figure(figsize=(15, 6))
plt.plot(df.index, df['Global_active_power'], label='Observed')
plt.plot(pd.date_range(df.index[-1], periods=30, freq='T'), forecast, label='Forecast')
plt.fill_between(pd.date_range(df.index[-1], periods=30, freq='T'), conf_int.iloc[:, 0], conf_int.iloc[:, 1], color='pink', alpha=0.3)
plt.xlabel('Date')
plt.ylabel('Global Active Power (kilowatts)')
plt.title('ARIMA Forecast')
plt.legend()
plt.show()
```

```
# Time series forecasting with SARIMA model
from statsmodels.tsa.statespace.sarimax import SARIMAX
import matplotlib.pyplot as plt

# Use a smaller subset of the data for testing
df_subset = df['Global_active_power'].iloc[-1000:]

# Ensure the subset has no NaN values
df_subset = df_subset.dropna()

# Define the model
sarima_model = SARIMAX(df_subset, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))

# Fit the model
sarima_result = sarima_model.fit(disp=False)

# Forecast
forecast = sarima_result.get_forecast(steps=30)
conf_int = forecast.conf_int()

# Plot the forecast
plt.figure(figsize=(15, 6))
plt.plot(df_subset.index, df_subset, label='Observed')
plt.plot(pd.date_range(df_subset.index[-1], periods=30, freq='T'), forecast.predicted_mean, label='Forecast')
plt.fill_between(pd.date_range(df_subset.index[-1], periods=30, freq='T'), conf_int.iloc[:, 0], conf_int.iloc[:, 1], color='pink', alpha
plt.xlabel('Date')
plt.ylabel('Global Active Power (kilowatts)')
plt.title('SARIMA Forecast')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-20a003433601> in <cell line: 6>()
      4 
      5 # Use a smaller subset of the data for testing
----> 6 df_subset = df['Global_active_power'].iloc[-1000:]
      7 
      8 # Ensure the subset has no NaN values

NameError: name 'df' is not defined
```

```
#LSTM model
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

# Scale the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df['Global_active_power'].values.reshape(-1, 1))

# Create sequences
def create_sequences(data, seq_length):
    sequences = []
    for i in range(len(data) - seq_length):
        sequences.append(data[i:i+seq_length])
    return np.array(sequences)

seq_length = 60
sequences = create_sequences(scaled_data, seq_length)

X = sequences[:, :-1]
y = sequences[:, -1]

# Split into train and test sets
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(seq_length-1, 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Make predictions
predictions = model.predict(X_test)

# Inverse scale the predictions
predictions = scaler.inverse_transform(predictions)

# Plot the results
plt.figure(figsize=(15, 6))
plt.plot(df.index[-len(y_test):], scaler.inverse_transform(y_test.reshape(-1, 1)), label='True')
plt.plot(df.index[-len(y_test):], predictions, label='Predicted')
plt.xlabel('Date')
plt.ylabel('Global Active Power (kilowatts)')
plt.title('LSTM Forecast')
plt.legend()
plt.show()
```

```
Epoch 1/10
10241/10241 [==============================] - 578s 56ms/step - loss: 9.8436e-04
Epoch 2/10
10241/10241 [==============================] - 561s 55ms/step - loss: 8.8156e-04
Epoch 3/10
10241/10241 [==============================] - 565s 55ms/step - loss: 8.7123e-04
Epoch 4/10
10241/10241 [==============================] - 572s 56ms/step - loss: 8.5612e-04
Epoch 5/10
10241/10241 [==============================] - 573s 56ms/step - loss: 7.9514e-04
Epoch 6/10
10241/10241 [==============================] - 568s 56ms/step - loss: 7.4218e-04
Epoch 7/10
 5330/10241 [==============>...............] - ETA: 4:30 - loss: 7.2315e-04
```