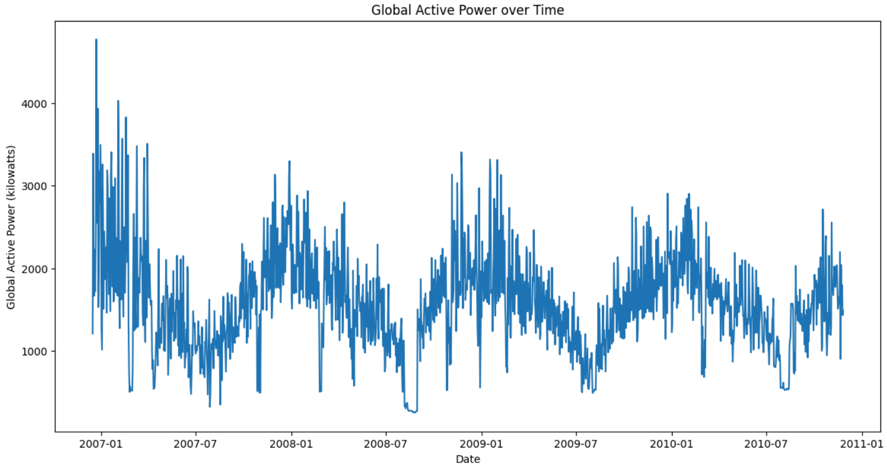
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

# Load the text file
df = pd.read_csv('household_power_consumption.txt', sep=';',parse_dates={'datetime': ['Date', 'Time']}, infer_datetime_format=True,low_memory

# Fill missing values with the mean
df['Global_active_power'] = pd.to_numeric(df['Global_active_power'], errors='coerce')
df['Global_active_power'].fillna(df['Global_active_power'].mean(), inplace=True)

# Set the datetime as the index
df.set_index('datetime', inplace=True)

# Resample the data to daily frequency
df_daily = df.resample('D').sum()
# Basic statistics
print(df_daily.describe())

# Check for missing values
print(df_daily.isnull().sum())

# Plotting the data
plt.figure(figsize=(14, 7))
plt.plot(df_daily.index, df_daily['Global_active_power'])
plt.title('Global Active Power over Time')
plt.xlabel('Date')
plt.ylabel('Global Active Power (kilowatts)')
plt.show()
```

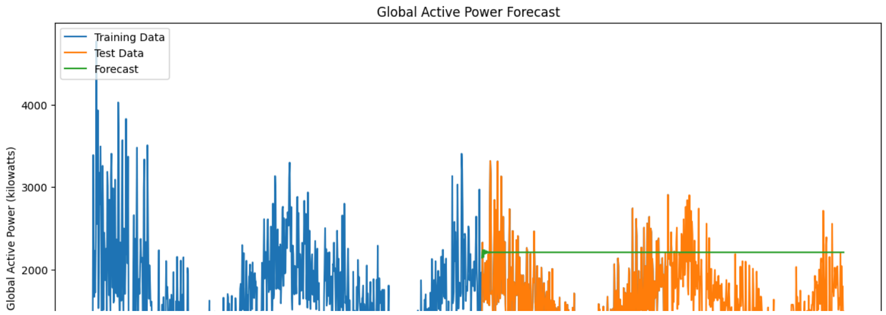
```
<ipython-input-15-3570c6b41cfa>:9: FutureWarning: The argument 'infer_datetime_format'
  df = pd.read_csv('household_power_consumption.txt', sep=';',parse_dates={'datetime':
<ipython-input-15-3570c6b41cfa>:9: UserWarning: Parsing dates in %d/%m/%Y %H:%M:%S form
  df = pd.read_csv('household_power_consumption.txt', sep=';',parse_dates={'datetime':
       Global_active_power  Global_reactive_power        Voltage  \
count          1442.000000            1442.000000    1442.000000
mean           1571.001338             175.815258  342266.507732
std             595.405647              51.998109   36707.752471
min             250.298000               0.000000       0.000000
25%            1191.182615             140.911500  345621.842500
50%            1559.085000             170.290000  346934.710000
75%            1889.859500             202.372500  348251.527500
max            4773.386000             417.834000  356306.410000

       Global_intensity  Sub_metering_1  Sub_metering_2  Sub_metering_3
count       1442.000000     1442.000000     1442.000000     1442.000000
mean        6576.681415     1594.407074     1845.375173     9178.340499
std         2559.505974     1587.840580     2089.590342     3787.898093
min            0.000000        0.000000        0.000000        0.000000
25%         4988.800000      555.500000      424.250000     6604.250000
50%         6510.300000     1109.000000      678.500000     9251.000000
75%         7953.350000     2196.750000     2712.750000    11708.500000
max        20200.400000    11178.000000    12109.000000    23743.000000
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
dtype: int64
```



```
# Define the training and testing periods
# Adjust these dates to have some overlap and avoid a gap
train_end_date = '2009-12-31'
test_start_date = '2009-01-01'

# Split the data
train = df_daily[:train_end_date]
test = df_daily[test_start_date:]

# Check if the test set is empty and handle the case
if test.empty:
    print("Warning: Test set is empty. Adjust the date range.")
else:
    # Fit an ARIMA model
    # You might need to experiment with different (p, d, q) orders
```

```
model = ARIMA(train['Global_active_power'], order=(5, 1, 0))
model_fit = model.fit()

# Forecasting
forecast = model_fit.forecast(steps=len(test))
test['Forecast'] = forecast.values

# Calculate MSE
mse = mean_squared_error(test['Global_active_power'], test['Forecast'])
print(f'Mean Squared Error: {mse}')

# Plot the results
plt.figure(figsize=(14, 7))
plt.plot(train.index, train['Global_active_power'], label='Training Data')
plt.plot(test.index, test['Global_active_power'], label='Test Data')
plt.plot(test.index, test['Forecast'], label='Forecast')
plt.legend(loc='upper left')
plt.title('Global Active Power Forecast')
plt.xlabel('Date')
plt.ylabel('Global Active Power (kilowatts)')
plt.show()
```

```
<ipython-input-19-40b94adc05a4>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  test['Forecast'] = forecast.values
Mean Squared Error: 689430.7539605086
```


Global Active Power Forecast
Training Data
Test Data
Forecast
Global Active Power (kilowatts)