In [1]:

```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
```

In [2]:

```python
df = pd.read_csv('dataset/train.csv')
submit = pd.read_csv('dataset/sample_submission.csv')
X = pd.read_csv('dataset/Weather.csv')
solar = pd.read_excel('dataset/solar_irradiance_full.xlsx')
# Convert the 'Timestamp' to a datetime object
df['Timestamp'] = pd.to_datetime(df['Timestamp'], format='%b %d, %Y %I%p')

# Extract the minimum and maximum dates from the original data
min_date = df['Timestamp'].min().date()
max_date = df['Timestamp'].max().date()

# Create a dictionary from the original data to map timestamps to baseline values
timestamp_baseline_map = dict(zip(df['Timestamp'], df['% Baseline']))

# Create a list to hold the new DataFrame rows
new_rows = []

# Iterate over each date from min_date to max_date
for date in pd.date_range(start=min_date, end=max_date, freq='D'):
    # Generate hourly timestamps for the specific date starting from 7 AM
    hourly_timestamps = pd.date_range(start=f'{date} 07:00:00', end=f'{date} 23:00:

    # Add the new rows with values from the map or a placeholder if missing
    for timestamp in hourly_timestamps:
        baseline_value = timestamp_baseline_map.get(timestamp, np.nan)  # Use 0 if
        new_rows.append([timestamp, baseline_value])

# Create the final DataFrame
final_df = pd.DataFrame(new_rows, columns=['Timestamp', '% Baseline'])

# Find the last timestamp in the original data
last_timestamp = df['Timestamp'].max()

# Filter the final_df to only include timestamps up to the last_timestamp in the or
df_filter = final_df[final_df['Timestamp'] <= last_timestamp]
```

```python
# Convert both columns to datetime objects
X['date_time'] = pd.to_datetime(X['date_time'], format='%m/%d/%Y %H:%M')
df_filter.loc[:, 'Timestamp'] = pd.to_datetime(df_filter['Timestamp'])

# Format both columns to have the same string representation
X.loc[:, 'date_time'] = X['date_time'].dt.strftime('%Y-%m-%d %H:%M:%S')
df_filter.loc[:, 'Timestamp'] = df_filter['Timestamp'].dt.strftime('%Y-%m-%d %H:%M:

# Combine the Year, Month, Day, Hour, and Minute columns to create a Timestamp colu
solar['Timestamp'] = pd.to_datetime(solar[['Year', 'Month', 'Day', 'Hour', 'Minute'
solar.drop(['Year','Month','Day','Hour','Minute'],axis=1,inplace=True)
```

```
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\1225792419.py:39: SettingWithCopy
Warning:
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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_filter.loc[:, 'Timestamp'] = pd.to_datetime(df_filter['Timestamp'])
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\1225792419.py:39: DeprecationWarn
ing: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values i
nplace instead of always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_filter.loc[:, 'Timestamp'] = pd.to_datetime(df_filter['Timestamp'])
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\1225792419.py:42: DeprecationWarn
ing: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values i
nplace instead of always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  X.loc[:, 'date_time'] = X['date_time'].dt.strftime('%Y-%m-%d %H:%M:%S')
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\1225792419.py:43: SettingWithCopy
Warning:
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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_filter.loc[:, 'Timestamp'] = df_filter['Timestamp'].dt.strftime('%Y-%m-%d %H:%
M:%S')
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\1225792419.py:43: DeprecationWarn
ing: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values i
nplace instead of always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_filter.loc[:, 'Timestamp'] = df_filter['Timestamp'].dt.strftime('%Y-%m-%d %H:%
M:%S')
```

In [3]:

```python
solar.head()
```

Out[3]:

	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Cloud Type	Dew Point	Solar Zenith Angle	Surface Albedo	Wind Speed
0	0.0	0.0	0.0	0.0	0.0	0.0	Probably Clear	-6	124.02	0.12	3.5
1	0.0	0.0	0.0	0.0	0.0	0.0	Probably Clear	-5	135.09	0.12	3.8
2	0.0	0.0	0.0	0.0	0.0	0.0	Probably Clear	-5	145.77	0.12	4.2
3	0.0	0.0	0.0	0.0	0.0	0.0	Probably Clear	-6	155.07	0.12	4.5
4	0.0	0.0	0.0	0.0	0.0	0.0	Probably Clear	-8	160.55	0.12	4.6

In [4]:

```
df_filter.head()
```

Out[4]:

	Timestamp	% Baseline
0	2014-01-01 07:00:00	0.0079
1	2014-01-01 08:00:00	0.1019
2	2014-01-01 09:00:00	0.3932
3	2014-01-01 10:00:00	0.5447
4	2014-01-01 11:00:00	0.5485

In [5]:

```
# Filter X based on matching timestamps in df_filter
X_filtered = X[X['date_time'].isin(df_filter['Timestamp'])]
X_filtered.drop(['moonrise','moonset','sunrise','sunset'],axis=1, inplace=True)

# Filter Solar
solar_filtered = solar[solar['Timestamp'].isin(df_filter['Timestamp'])]
```

```
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\1412383839.py:3: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  X_filtered.drop(['moonrise','moonset','sunrise','sunset'],axis=1, inplace=True)
```

In [6]:

```
# Perform the merge/join based on the timestamp columns
join = pd.merge(df_filter, X_filtered, left_on='Timestamp', right_on='date_time')
# Ensure both 'Timestamp' columns are in datetime format
join['Timestamp'] = pd.to_datetime(join['Timestamp'])
solar_filtered['Timestamp'] = pd.to_datetime(solar_filtered['Timestamp'])

# Perform the merge on the 'Timestamp' column
```

```
join = pd.merge(join, solar_filtered, on='Timestamp', how='inner')
# Drop the redundant 'date_time' column after merge if desired
join = join.drop(columns=['date_time','Timestamp'])
join_fil = join.dropna()
join_fil.head()
```

```
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\2765218598.py:5: SettingWithCopyW
arning:
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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  solar_filtered['Timestamp'] = pd.to_datetime(solar_filtered['Timestamp'])
```

Out[6]:

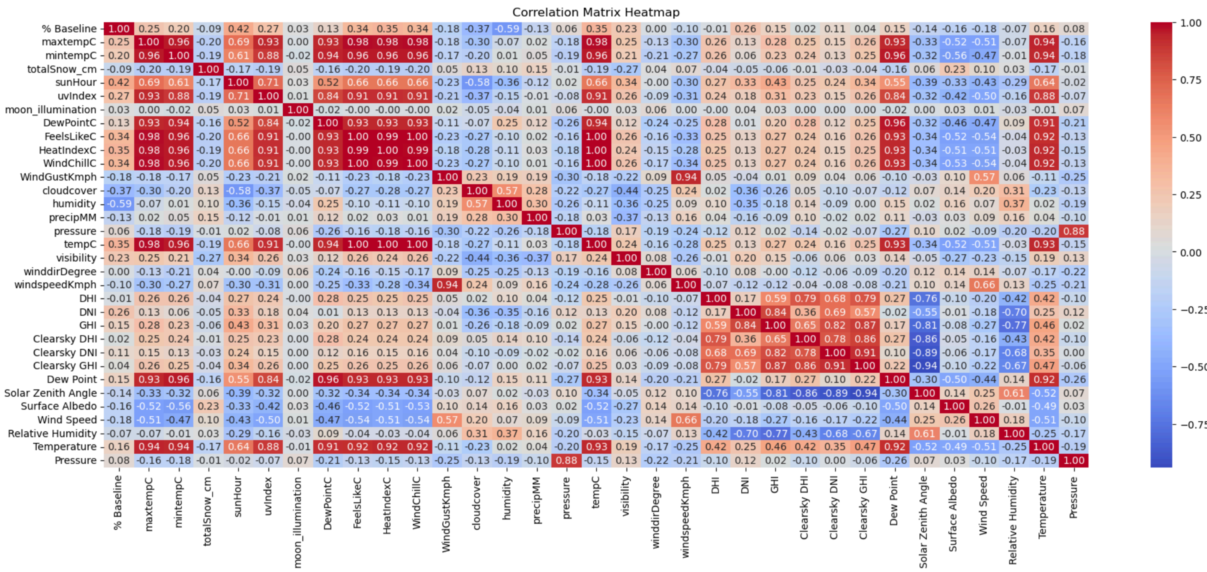
	% Baseline	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	moon_illumination
0	0.0079	-3	-6	0.0	8.7	2	1
1	0.1019	-3	-6	0.0	8.7	2	1
2	0.3932	-3	-6	0.0	8.7	2	1
3	0.5447	-3	-6	0.0	8.7	2	1
4	0.5485	-3	-6	0.0	8.7	2	1

5 rows × 34 columns

In [7]:

```
# Calculate the correlation matrix
correlation_matrix = join_fil.corr()

# Create a heatmap using seaborn
plt.figure(figsize=(22, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')

# Set the title and display the plot
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\3182376918.py:2: FutureWarning: T
he default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of numer
ic_only to silence this warning.
  correlation_matrix = join_fil.corr()
```



In [8]:
```python
baseline_correlation = correlation_matrix['% Baseline']

# Determine which columns to keep (correlation >= 0.2)
columns_to_keep = baseline_correlation[abs(baseline_correlation) >= 0.1].index
exported = join[columns_to_keep]

# Drop columns with correlation < 0.2
filtered_df = join_fil[columns_to_keep]

# Display the filtered DataFrame
filtered_df.head()
```

Out[8]:

	% Baseline	maxtempC	mintempC	sunHour	uvIndex	DewPointC	FeelsLikeC	HeatIndexC
0	0.0079	-3	-6	8.7	2	-14	-13	-5
1	0.1019	-3	-6	8.7	2	-14	-12	-5
2	0.3932	-3	-6	8.7	2	-14	-11	-4
3	0.5447	-3	-6	8.7	2	-14	-10	-4
4	0.5485	-3	-6	8.7	2	-14	-10	-3

5 rows × 24 columns

In [16]:
```python
sub = pd.read_csv('dataset/sample_submission.csv')
sub['Timestamp'] = pd.to_datetime(sub['Timestamp'])
sub.head()
```

Out[16]:

	Timestamp	% Baseline
0	2017-10-01 06:00:00	0.0005
1	2017-10-01 07:00:00	0.0005
2	2017-10-01 08:00:00	0.0005
3	2017-10-01 09:00:00	0.0005
4	2017-10-01 10:00:00	0.0005

In [15]:
```
df_filter.tail()
```

Out[15]:

	Timestamp	% Baseline
23263	2017-09-30 14:00:00	0.1846
23264	2017-09-30 15:00:00	0.0711
23265	2017-09-30 16:00:00	0.0560
23266	2017-09-30 17:00:00	0.0182
23267	2017-09-30 18:00:00	0.0004

In [17]:
```
X.tail()
```

Out[17]:

	date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	moon_illumina
35059	2017-12-31 19:00:00	-9	-13	0.2	8.7	1	
35060	2017-12-31 20:00:00	-9	-13	0.2	8.7	1	
35061	2017-12-31 21:00:00	-9	-13	0.2	8.7	1	
35062	2017-12-31 22:00:00	-9	-13	0.2	8.7	1	
35063	2017-12-31 23:00:00	-9	-13	0.2	8.7	1	

5 rows × 24 columns

In [46]:
```
df_filter['Timestamp'] = pd.to_datetime(df_filter['Timestamp'])
X['date_time'] = pd.to_datetime(X['date_time'])
full_timestamp_range = pd.date_range(start=df_filter['Timestamp'].min(),
```

```
                                     end=X['date_time'].max(),
                                     freq='H')  # Assuming hourly frequency
df_extended = pd.DataFrame(full_timestamp_range, columns=['Timestamp'])
df_extended = pd.merge(df_extended, df_filter, on='Timestamp', how='left')
df_extended = pd.merge(df_extended, X, left_on='Timestamp', right_on='date_time', h
df_extended = pd.merge(df_extended, solar, on='Timestamp', how='inner')
df_extended.head()
```

```
C:\Users\NAUFAL\AppData\Local\Temp\ipykernel_30448\1624733487.py:1: SettingWithCopyW
arning:
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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_filter['Timestamp'] = pd.to_datetime(df_filter['Timestamp'])
```

Out[46]:

	Timestamp	% Baseline	date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvInd
0	2014-01-01 07:00:00	0.0079	2014-01-01 07:00:00	-3	-6	0.0	8.7	
1	2014-01-01 08:00:00	0.1019	2014-01-01 08:00:00	-3	-6	0.0	8.7	
2	2014-01-01 09:00:00	0.3932	2014-01-01 09:00:00	-3	-6	0.0	8.7	
3	2014-01-01 10:00:00	0.5447	2014-01-01 10:00:00	-3	-6	0.0	8.7	
4	2014-01-01 11:00:00	0.5485	2014-01-01 11:00:00	-3	-6	0.0	8.7	

5 rows × 40 columns

In [51]:

```
columns_list = list(columns_to_keep)  # Convert Index to a list
columns_list.insert(0, 'Timestamp')   # Insert 'Timestamp' at the beginning
columns_list = pd.Index(columns_list)
```

In [53]:

```
columns_list
```

Out[53]:

```
Index(['Timestamp', '% Baseline', 'maxtempC', 'mintempC', 'sunHour', 'uvIndex',
       'DewPointC', 'FeelsLikeC', 'HeatIndexC', 'WindChillC', 'WindGustKmph',
       'cloudcover', 'humidity', 'precipMM', 'tempC', 'visibility',
       'windspeedKmph', 'DNI', 'GHI', 'Clearsky DNI', 'Dew Point',
       'Solar Zenith Angle', 'Surface Albedo', 'Wind Speed', 'Temperature'],
      dtype='object')
```

In [54]:
```
aa = df_extended[columns_list]
aa.columns
```

Out[54]:
```
Index(['Timestamp', '% Baseline', 'maxtempC', 'mintempC', 'sunHour', 'uvIndex',
       'DewPointC', 'FeelsLikeC', 'HeatIndexC', 'WindChillC', 'WindGustKmph',
       'cloudcover', 'humidity', 'precipMM', 'tempC', 'visibility',
       'windspeedKmph', 'DNI', 'GHI', 'Clearsky DNI', 'Dew Point',
       'Solar Zenith Angle', 'Surface Albedo', 'Wind Speed', 'Temperature'],
      dtype='object')
```

In [55]:
```
aa.shape
```

Out[55]:
```
(35057, 25)
```

In [56]:
```
aa.to_csv('train_clean_v2.csv')
```

In [136…
```
# Assuming `filtered_df` is your DataFrame and '% Baseline' is the target variable
X = filtered_df.drop(columns=['% Baseline'])
y = filtered_df['% Baseline']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features (important for neural networks)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [137…
```
ann = Sequential()
unit = (filtered_df.shape[1]-1)*20
ann.add(Dense(units = unit, activation = "relu"))
ann.add(Dense(units = unit, activation = "relu"))
ann.add(Dense(units = unit, activation = "relu"))
ann.add(Dense(units = unit, activation = "relu"))
ann.add(Dense(units = 1))
ann.compile(optimizer = "adam", loss = "mean_squared_error")
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=20,restore_best
history = ann.fit(X_train, y_train, batch_size = 32, epochs = 150, validation_data=
```

```
Epoch 1/150
421/421 15s 24ms/step - loss: 0.0429 - val_loss: 0.0170
Epoch 2/150
421/421 10s 25ms/step - loss: 0.0161 - val_loss: 0.0171
Epoch 3/150
421/421 20s 24ms/step - loss: 0.0140 - val_loss: 0.0137
Epoch 4/150
421/421 11s 27ms/step - loss: 0.0131 - val_loss: 0.0136
Epoch 5/150
421/421 12s 29ms/step - loss: 0.0120 - val_loss: 0.0134
Epoch 6/150
421/421 20s 27ms/step - loss: 0.0116 - val_loss: 0.0120
Epoch 7/150
421/421 10s 24ms/step - loss: 0.0107 - val_loss: 0.0122
Epoch 8/150
421/421 9s 21ms/step - loss: 0.0102 - val_loss: 0.0115
Epoch 9/150
421/421 10s 23ms/step - loss: 0.0094 - val_loss: 0.0132
Epoch 10/150
421/421 11s 24ms/step - loss: 0.0094 - val_loss: 0.0111
Epoch 11/150
421/421 11s 25ms/step - loss: 0.0087 - val_loss: 0.0127
Epoch 12/150
421/421 10s 24ms/step - loss: 0.0087 - val_loss: 0.0107
Epoch 13/150
421/421 8s 19ms/step - loss: 0.0072 - val_loss: 0.0095
Epoch 14/150
421/421 8s 19ms/step - loss: 0.0072 - val_loss: 0.0112
Epoch 15/150
421/421 10s 23ms/step - loss: 0.0069 - val_loss: 0.0098
Epoch 16/150
421/421 10s 23ms/step - loss: 0.0061 - val_loss: 0.0099
Epoch 17/150
421/421 10s 23ms/step - loss: 0.0059 - val_loss: 0.0110
Epoch 18/150
421/421 10s 23ms/step - loss: 0.0063 - val_loss: 0.0088
Epoch 19/150
421/421 10s 22ms/step - loss: 0.0058 - val_loss: 0.0092
Epoch 20/150
421/421 9s 22ms/step - loss: 0.0054 - val_loss: 0.0098
Epoch 21/150
421/421 9s 22ms/step - loss: 0.0048 - val_loss: 0.0096
Epoch 22/150
421/421 11s 24ms/step - loss: 0.0046 - val_loss: 0.0089
Epoch 23/150
421/421 11s 25ms/step - loss: 0.0044 - val_loss: 0.0087
Epoch 24/150
421/421 9s 22ms/step - loss: 0.0041 - val_loss: 0.0088
Epoch 25/150
421/421 11s 26ms/step - loss: 0.0040 - val_loss: 0.0084
Epoch 26/150
421/421 11s 26ms/step - loss: 0.0036 - val_loss: 0.0085
Epoch 27/150
421/421 10s 23ms/step - loss: 0.0040 - val_loss: 0.0084
Epoch 28/150
421/421 9s 22ms/step - loss: 0.0035 - val_loss: 0.0081
```

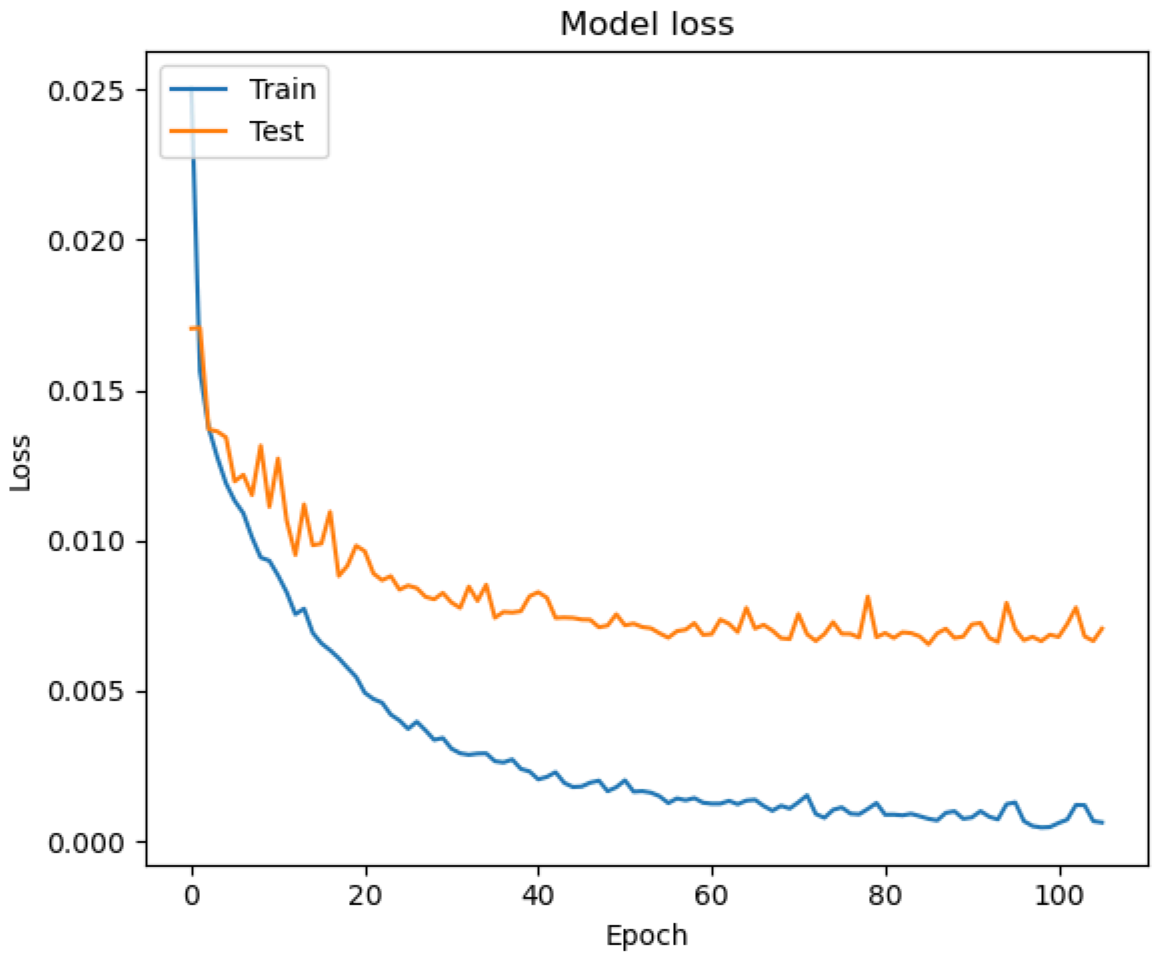
```
Epoch 29/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0035 - val_loss: 0.0080
Epoch 30/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0033 - val_loss: 0.0083
Epoch 31/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - loss: 0.0030 - val_loss: 0.0079
Epoch 32/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0028 - val_loss: 0.0078
Epoch 33/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 11s 24ms/step - loss: 0.0025 - val_loss: 0.0085
Epoch 34/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - loss: 0.0029 - val_loss: 0.0080
Epoch 35/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0028 - val_loss: 0.0085
Epoch 36/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0026 - val_loss: 0.0074
Epoch 37/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - loss: 0.0027 - val_loss: 0.0076
Epoch 38/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - loss: 0.0027 - val_loss: 0.0076
Epoch 39/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - loss: 0.0021 - val_loss: 0.0076
Epoch 40/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - loss: 0.0022 - val_loss: 0.0081
Epoch 41/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0020 - val_loss: 0.0083
Epoch 42/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0021 - val_loss: 0.0081
Epoch 43/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - loss: 0.0024 - val_loss: 0.0074
Epoch 44/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0019 - val_loss: 0.0074
Epoch 45/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0017 - val_loss: 0.0074
Epoch 46/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0016 - val_loss: 0.0074
Epoch 47/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0018 - val_loss: 0.0074
Epoch 48/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0020 - val_loss: 0.0071
Epoch 49/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - loss: 0.0016 - val_loss: 0.0072
Epoch 50/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0018 - val_loss: 0.0075
Epoch 51/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0018 - val_loss: 0.0072
Epoch 52/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - loss: 0.0016 - val_loss: 0.0072
Epoch 53/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 0.0018 - val_loss: 0.0071
Epoch 54/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0016 - val_loss: 0.0071
Epoch 55/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - loss: 0.0015 - val_loss: 0.0069
Epoch 56/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0012 - val_loss: 0.0068
```

```
Epoch 57/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - loss: 0.0013 - val_loss: 0.0070
Epoch 58/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0012 - val_loss: 0.0070
Epoch 59/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0015 - val_loss: 0.0073
Epoch 60/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0013 - val_loss: 0.0069
Epoch 61/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 20ms/step - loss: 0.0012 - val_loss: 0.0069
Epoch 62/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - loss: 0.0012 - val_loss: 0.0074
Epoch 63/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - loss: 0.0012 - val_loss: 0.0072
Epoch 64/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - loss: 0.0012 - val_loss: 0.0070
Epoch 65/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - loss: 0.0012 - val_loss: 0.0078
Epoch 66/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - loss: 0.0015 - val_loss: 0.0071
Epoch 67/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - loss: 0.0011 - val_loss: 0.0072
Epoch 68/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - loss: 0.0010 - val_loss: 0.0070
Epoch 69/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - loss: 0.0011 - val_loss: 0.0067
Epoch 70/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - loss: 0.0010 - val_loss: 0.0067
Epoch 71/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - loss: 0.0012 - val_loss: 0.0076
Epoch 72/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 11s 26ms/step - loss: 0.0017 - val_loss: 0.0069
Epoch 73/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 9.2031e-04 - val_loss: 0.0067
Epoch 74/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 7.0434e-04 - val_loss: 0.0069
Epoch 75/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 8.3645e-04 - val_loss: 0.0073
Epoch 76/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 0.0012 - val_loss: 0.0069
Epoch 77/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 9.4853e-04 - val_loss: 0.0069
Epoch 78/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - loss: 9.2484e-04 - val_loss: 0.0068
Epoch 79/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 8.2835e-04 - val_loss: 0.0081
Epoch 80/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 0.0015 - val_loss: 0.0068
Epoch 81/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 8.3470e-04 - val_loss: 0.0069
Epoch 82/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 8.7553e-04 - val_loss: 0.0068
Epoch 83/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 8.8971e-04 - val_loss: 0.0069
Epoch 84/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 11s 21ms/step - loss: 7.7441e-04 - val_loss: 0.0069
```

```
Epoch 85/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 8.1930e-04 - val_loss: 0.0068
Epoch 86/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 7.8230e-04 - val_loss: 0.0065
Epoch 87/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 6.5902e-04 - val_loss: 0.0069
Epoch 88/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 7.8525e-04 - val_loss: 0.0071
Epoch 89/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0011 - val_loss: 0.0068
Epoch 90/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - loss: 7.9105e-04 - val_loss: 0.0068
Epoch 91/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - loss: 6.4908e-04 - val_loss: 0.0072
Epoch 92/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - loss: 0.0011 - val_loss: 0.0073
Epoch 93/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 8.8328e-04 - val_loss: 0.0068
Epoch 94/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 6.7517e-04 - val_loss: 0.0066
Epoch 95/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 8.8136e-04 - val_loss: 0.0079
Epoch 96/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 0.0015 - val_loss: 0.0070
Epoch 97/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 6.6023e-04 - val_loss: 0.0067
Epoch 98/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 4.5939e-04 - val_loss: 0.0068
Epoch 99/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - loss: 4.5979e-04 - val_loss: 0.0067
Epoch 100/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 4.4530e-04 - val_loss: 0.0069
Epoch 101/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 5.2688e-04 - val_loss: 0.0068
Epoch 102/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 11s 22ms/step - loss: 6.8092e-04 - val_loss: 0.0072
Epoch 103/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 9.6327e-04 - val_loss: 0.0078
Epoch 104/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 0.0015 - val_loss: 0.0068
Epoch 105/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 6.9048e-04 - val_loss: 0.0067
Epoch 106/150
421/421 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - loss: 5.1529e-04 - val_loss: 0.0071
```

In [138…

```python
# Plot training & validation loss values
plt.figure(figsize=(6, 5))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

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



In [139…
```
pred = ann.predict(X_test)
tpred = ann.predict(X_train)
print('Train R2 Score : ',r2_score(y_train,tpred))
print('Test R2 Score : ',r2_score(y_test,pred))
```

```
106/106 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
421/421 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Train R2 Score :  0.990520082179919
Test R2 Score :  0.9017448124976732
```

In [140…
```
0.90-0.895
```

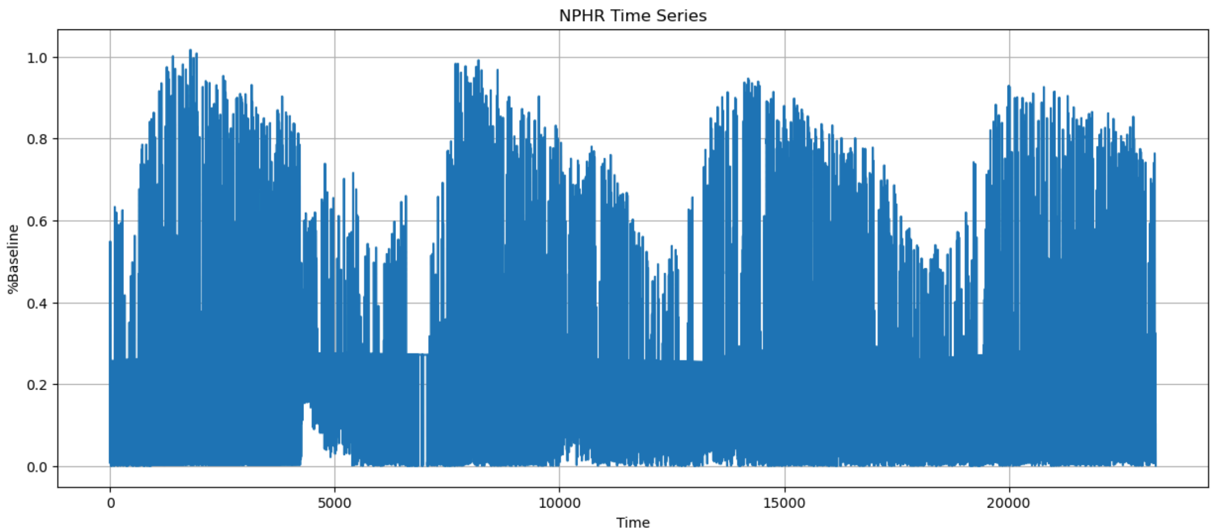
Out[140…
```
0.0050000000000000044
```

In [134…
```
# Save the entire model
#ann.save('ann_model.h5')
```

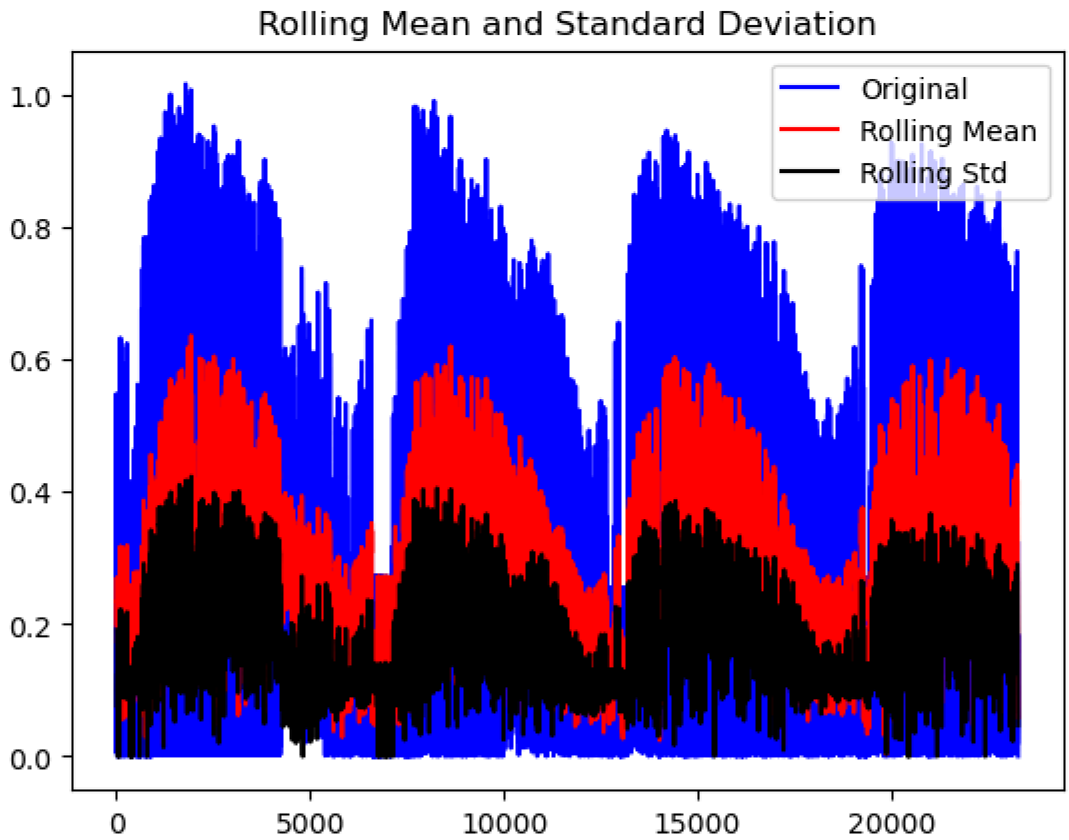
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

In [56]:
```
plt.figure(figsize=(15,6))
plt.grid(True)
plt.xlabel('Time')
plt.ylabel('%Baseline')
```

```
plt.plot(df_filter['% Baseline'])
plt.title('NPHR Time Series')
plt.show()
```



In [57]:
```
#Test for stationarity
def test_stationarity(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()
    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    print("Results of Dickey Fuller test")
    adft = adfuller(timeseries,autolag='AIC')
    # output for dft will give us without defining what the values are.
    #hence we manually write what values does it explains using a for loop
    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags us
    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values
    print(output)

test_stationarity(df_filter['% Baseline'])
```

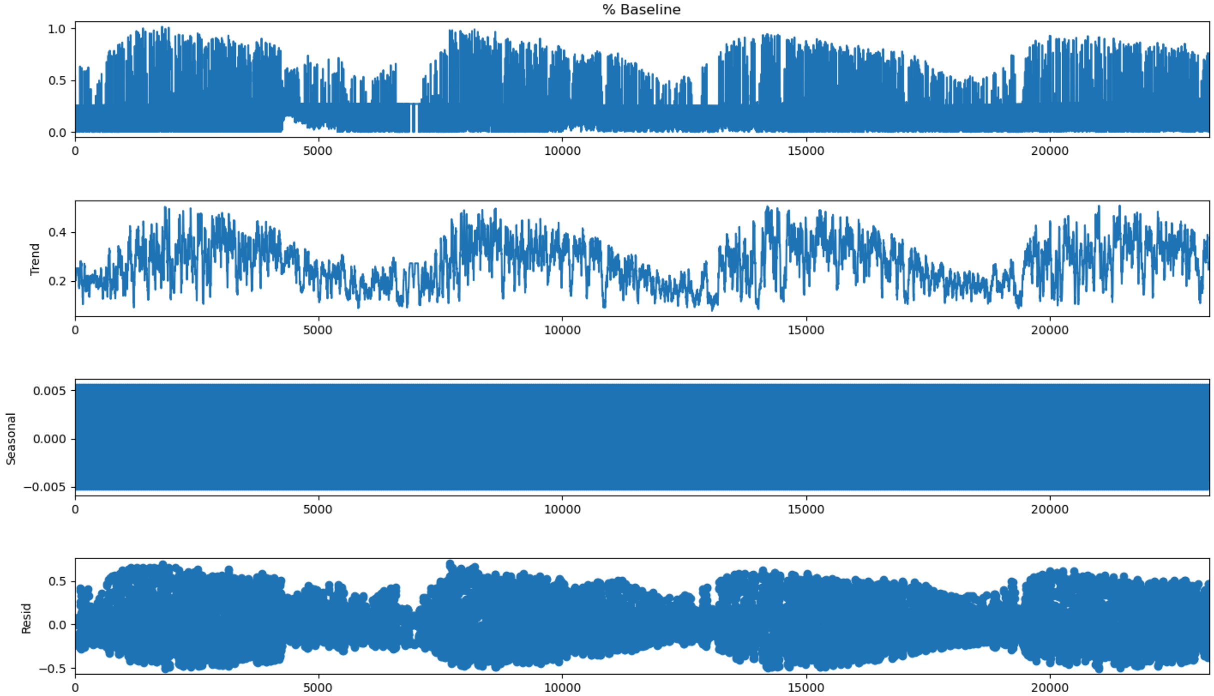



```
Results of Dickey Fuller test
Test Statistics                 -1.269716e+01
p-value                          1.099764e-23
No. of lags used                 4.700000e+01
Number of observations used      2.322000e+04
critical value (1%)             -3.430632e+00
critical value (5%)             -2.861664e+00
critical value (10%)            -2.566836e+00
dtype: float64
```

In [58]:

```
result = seasonal_decompose(df_filter['% Baseline'], model='additive',period=30)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(16, 9)
```

```
<Figure size 640x480 with 0 Axes>
```



In [18]:

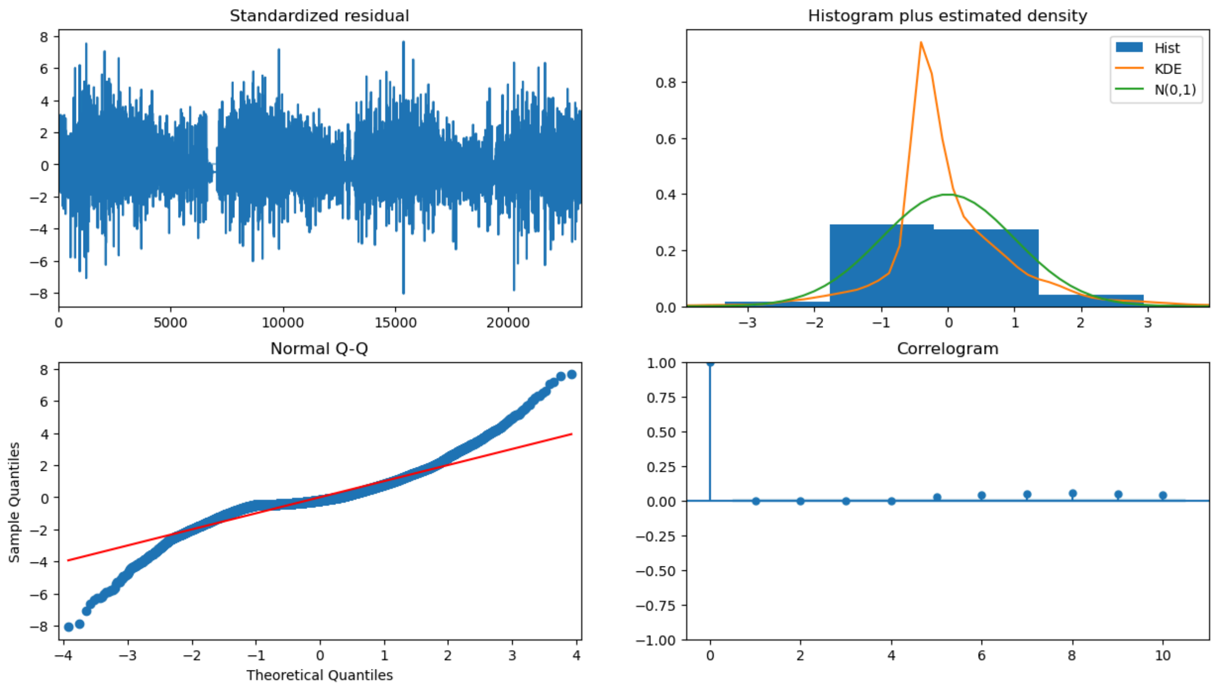
```
XX = X_filtered[['WindChillC','WindGustKmph','cloudcover']]
model_autoARIMA = auto_arima(df_filter['% Baseline'], start_p=0, start_q=0,
                      test='adf',       # use adftest to find             optimal '
                      max_p=3, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True,
                      exogenous= XX)
print(model_autoARIMA.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=12924.902, Time=1.09 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=-40558.901, Time=0.75 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=-13745.822, Time=2.16 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=-51223.708, Time=1.48 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=-51230.933, Time=2.34 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=-51230.235, Time=10.96 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=-51230.333, Time=3.35 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=-53351.572, Time=5.90 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=-53093.955, Time=1.75 sec
 ARIMA(3,0,1)(0,0,0)[0] intercept   : AIC=-53414.515, Time=29.78 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=-53424.572, Time=13.09 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=-48823.340, Time=6.41 sec
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=-53515.518, Time=24.85 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=-50926.113, Time=16.13 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=-53460.738, Time=48.57 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=-53508.371, Time=30.87 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=-51914.036, Time=18.62 sec
 ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=-53511.728, Time=30.65 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=-51237.738, Time=5.49 sec

Best model:  ARIMA(2,0,2)(0,0,0)[0] intercept
Total fit time: 254.365 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                23268
Model:               SARIMAX(2, 0, 2)   Log Likelihood               26763.759
Date:                Tue, 20 Aug 2024   AIC                         -53515.518
Time:                        19:00:53   BIC                         -53467.189
Sample:                             0   HQIC                        -53499.821
                              - 23268
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0249      0.001     31.475      0.000       0.023       0.026
ar.L1          1.6595      0.008    199.448      0.000       1.643       1.676
ar.L2         -0.7806      0.007   -109.364      0.000      -0.795      -0.767
ma.L1         -0.2444      0.009    -25.802      0.000      -0.263      -0.226
ma.L2         -0.0801      0.007    -11.118      0.000      -0.094      -0.066
sigma2         0.0059   3.57e-05    164.100      0.000       0.006       0.006
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             32858.16
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               0.95   Skew:                             0.50
Prob(H) (two-sided):                  0.02   Kurtosis:                         8.73
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

In [19]:

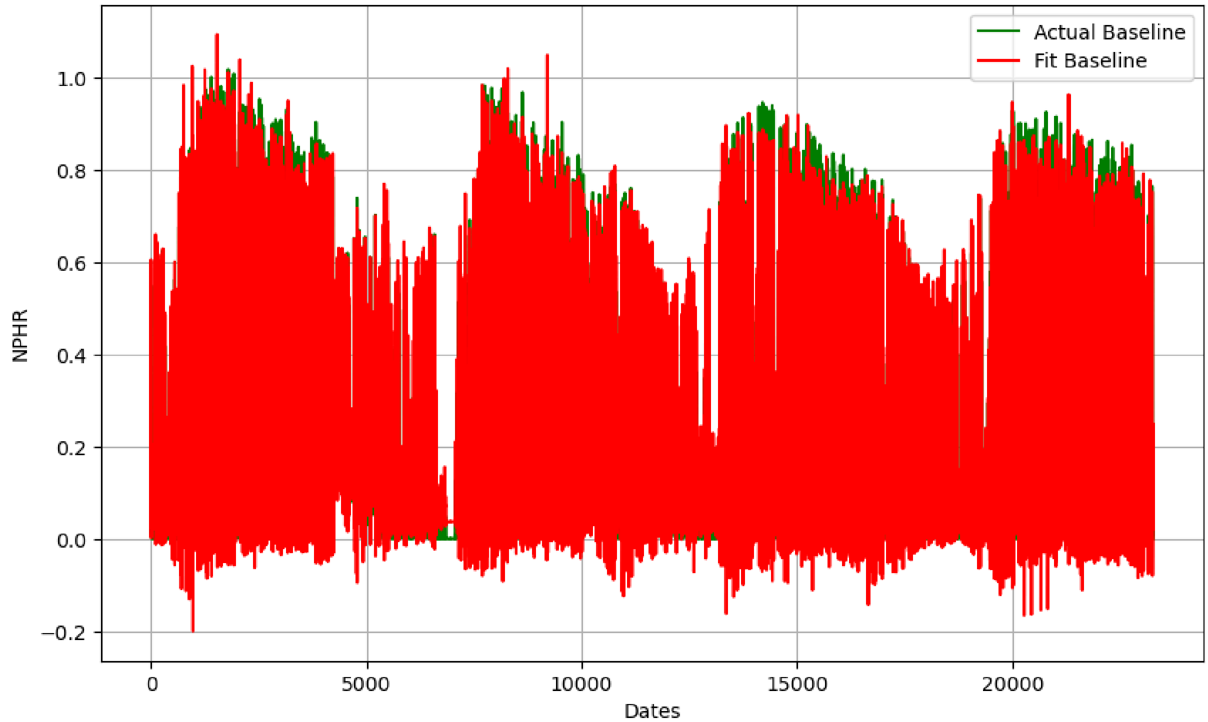
```
def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean(np.abs((actual - pred) / actual)) * 100
```

In [20]:
```python
model_autoARIMA.plot_diagnostics(figsize=(15,8))
plt.show()
```



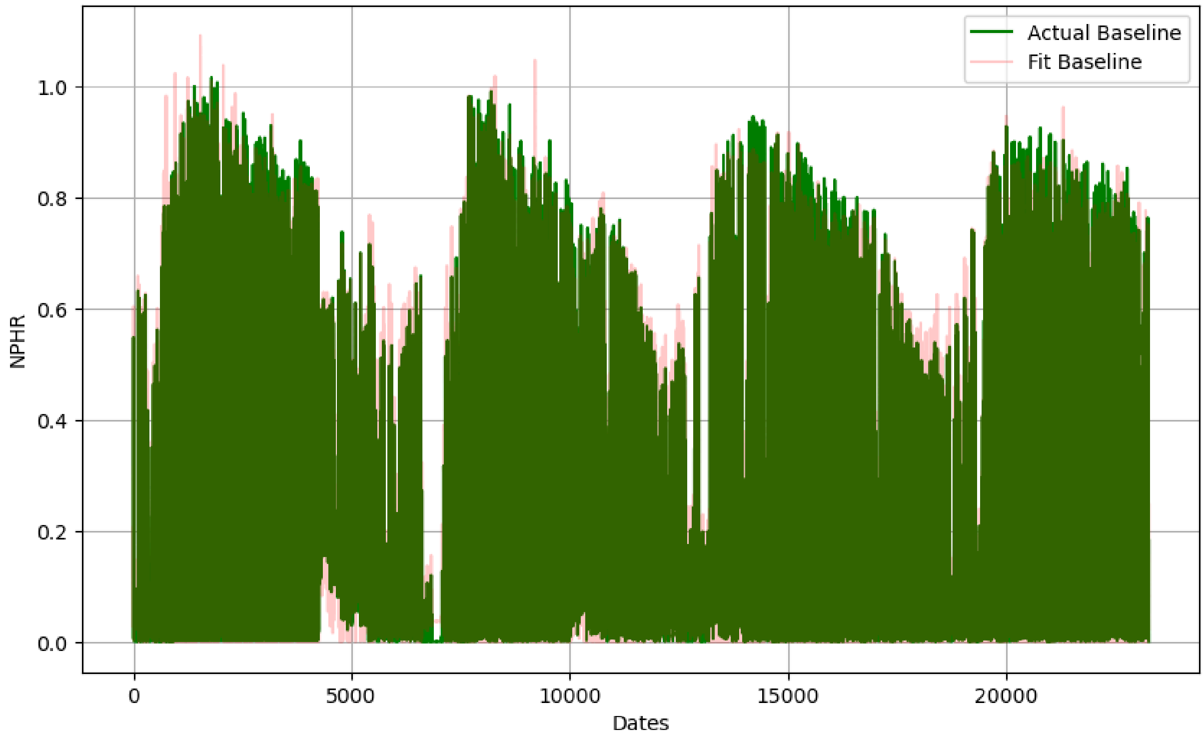
In [32]:
```python
fit_bs = model_autoARIMA.fittedvalues()
plt.figure(figsize=(10,6))
plt.grid(True)
plt.xlabel('Dates')
plt.ylabel('NPHR')
plt.plot(df_filter['% Baseline'], 'green', label='Actual Baseline')
plt.plot(fit_bs, 'red', label='Fit Baseline')
plt.legend()
print(np.sqrt(mean_squared_error(df_filter['% Baseline'], fit_bs)))
```

```
0.07660454589992081
```



In [33]:

```
# Replace negative values in fit_bs with 0
fit_bs = np.where(fit_bs < 0, 0, fit_bs)

# Plot the results again
plt.figure(figsize=(10,6))
plt.grid(True)
plt.xlabel('Dates')
plt.ylabel('NPHR')
plt.plot(df_filter['% Baseline'], 'green', label='Actual Baseline')
plt.plot(fit_bs, 'red', label='Fit Baseline', alpha=0.2)
plt.legend()
plt.show()
print(np.sqrt(mean_squared_error(df_filter['% Baseline'], fit_bs)))
```




```
0.07564483165475702
```