```python
# Import Libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split

# Load Dataset (Upload in Google Colab)
from google.colab import files
uploaded = files.upload()

# Read Dataset
df = pd.read_csv(next(iter(uploaded)))  # Auto-detect uploaded filename

# Display first few rows
print("Dataset Preview:")
print(df.head())

# Strip whitespace from column names to avoid errors
df.columns = df.columns.str.strip()

# Check if 'demand' exists in the dataset
if 'demand' not in df.columns:
    print("Error: 'demand' column not found. Check the dataset for exact column names!")
    print(f"Available columns: {df.columns}")
else:
    print("'demand' column found, proceeding with preprocessing.")

# Drop unnecessary columns
df.drop(columns=['id', 'timestamp'], errors='ignore', inplace=True)  # 'errors=ignore' prevents KeyErrors

# Handle Missing Values
df.fillna(df.median(), inplace=True)

# Separate Features and Target Variable
X = df.drop(columns=['demand'], errors='ignore')  # Features
y = df['demand']  # Target Variable

# Standardize Features (Required for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA (Retain 95% Variance)
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

# Alternative: Feature Selection (Remove Low-Variance Features)
selector = VarianceThreshold(threshold=0.01)
X_selected = selector.fit_transform(X_scaled)

# Choose One: PCA or Feature Selection
X_final = X_pca  # Use X_selected if using feature selection instead
```

```
# Splitting Data into Train & Test Sets
X_train, X_test, y_train, y_test = train_test_split(X_final, y, test_size=0.2, random_state=42)

# Build ANN Model
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Output Layer for Regression
])

# Compile Model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train Model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))

# Evaluate Model
test_loss, test_mae = model.evaluate(X_test, y_test)
print(f"Test MAE: {test_mae:.2f}")
```

Choose Files gridwatch.csv

- **gridwatch.csv**(text/csv) - 6799951 bytes, last modified: 3/6/2025 - 100% done

```
Saving gridwatch.csv to gridwatch.csv
Dataset Preview:
        id            timestamp  demand  frequency  coal  nuclear  ccgt  \
0  1375187  2024-07-01 00:00:31   23710  49.955002     0     4684  2808
1  1375188  2024-07-01 00:05:32   23710  49.955002     0     4684  2808
2  1375189  2024-07-01 00:10:32   23710  49.955002     0     4684  2808
3  1375190  2024-07-01 00:15:31   23710  49.955002     0     4684  2808
4  1375191  2024-07-01 00:20:32   23710  49.955002     0     4684  2808

   wind  pumped  hydro  ...  irish_ict  ew_ict  nemo  other  \
0  8566       0    267  ...       -452    -528    23    298
1  8566       0    267  ...       -452    -528    23    298
2  8566       0    267  ...       -452    -528    23    298
3  8566       0    267  ...       -452    -528    23    298
4  8566       0    267  ...       -452    -528    23    298

   north_south  scotland_england  ifa2  intelec_ict   nsl  vkl_ict
0            0                 0   992          996  1399      368
1            0                 0   992          996  1399      368
2            0                 0   992          996  1399      368
3            0                 0   992          996  1399      368
4            0                 0   992          996  1399      368

[5 rows x 26 columns]
'demand' column found, proceeding with preprocessing.
Epoch 1/100
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pre
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 397862816.0000 - mae: 16022.7773 - val_loss: 7499226.5000 - val_mae: 2121.8467
Epoch 2/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 6829155.0000 - mae: 1981.1975 - val_loss: 4866866.0000 - val_mae: 1694.0947
Epoch 3/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 4544124.0000 - mae: 1613.3524 - val_loss: 3764654.2500 - val_mae: 1490.3248
Epoch 4/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 3574831.0000 - mae: 1428.1865 - val_loss: 3085724.5000 - val_mae: 1327.1970
Epoch 5/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 3007228.5000 - mae: 1299.6368 - val_loss: 2706512.0000 - val_mae: 1258.5735
Epoch 6/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 2721373.0000 - mae: 1229.5883 - val_loss: 2559347.5000 - val_mae: 1191.4304
Epoch 7/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 2548479.0000 - mae: 1154.2201 - val_loss: 2233379.0000 - val_mae: 1105.1646
Epoch 8/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 2279668.2500 - mae: 1089.5607 - val_loss: 2107375.5000 - val_mae: 1081.7981
Epoch 9/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 2095877.7500 - mae: 1063.2750 - val_loss: 1944911.5000 - val_mae: 1035.4016
Epoch 10/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 2003131.8750 - mae: 1022.6491 - val_loss: 1860181.1250 - val_mae: 998.7943
Epoch 11/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 1925513.1250 - mae: 983.6747 - val_loss: 1724226.2500 - val_mae: 965.6026
Epoch 12/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 1743799.0000 - mae: 955.1795 - val_loss: 1607858.0000 - val_mae: 962.2882
Epoch 13/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 1614301.8750 - mae: 919.3728 - val_loss: 1540870.3750 - val_mae: 904.4214
Epoch 14/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 1545116.0000 - mae: 885.1542 - val_loss: 1407900.3750 - val_mae: 875.4192
Epoch 15/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 1390802.8750 - mae: 858.0029 - val_loss: 1328346.7500 - val_mae: 842.9573
Epoch 16/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 1544089.7500 - mae: 845.1821 - val_loss: 1254008.6250 - val_mae: 819.1450
Epoch 17/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 1342175.0000 - mae: 819.3183 - val_loss: 1152412.3750 - val_mae: 790.0506
```

```
Epoch 18/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - loss: 1181496.3750 - mae: 789.9255 - val_loss: 1105586.7500 - val_mae: 775.9633
Epoch 19/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 1309493.3750 - mae: 778.8516 - val_loss: 1085394.8750 - val_mae: 759.4750
Epoch 20/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 1213266.8750 - mae: 745.8705 - val_loss: 1029821.6250 - val_mae: 745.7785
Epoch 21/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 1083642.2500 - mae: 725.7601 - val_loss: 977554.5000 - val_mae: 722.7173
Epoch 22/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 1011226.6250 - mae: 709.4836 - val_loss: 965033.8125 - val_mae: 724.4973
Epoch 23/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 1001512.7500 - mae: 694.7629 - val_loss: 966306.5000 - val_mae: 737.8350
Epoch 24/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 868158.1875 - mae: 678.3546 - val_loss: 896246.5000 - val_mae: 692.6180
Epoch 25/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - loss: 1011845.8125 - mae: 671.0162 - val_loss: 819714.2500 - val_mae: 663.0778
Epoch 26/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 856507.3125 - mae: 648.3056 - val_loss: 772980.0000 - val_mae: 639.4173
Epoch 27/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 848911.4375 - mae: 641.9761 - val_loss: 770740.8125 - val_mae: 637.7764
Epoch 28/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 927407.7500 - mae: 640.1866 - val_loss: 751367.6250 - val_mae: 630.3107
Epoch 29/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 856192.5000 - mae: 630.4323 - val_loss: 735898.7500 - val_mae: 625.9243
Epoch 30/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 757626.1875 - mae: 611.1766 - val_loss: 770428.8750 - val_mae: 631.0770
Epoch 31/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 791497.7500 - mae: 606.8832 - val_loss: 685664.1875 - val_mae: 596.0843
Epoch 32/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 747137.6250 - mae: 596.0504 - val_loss: 683336.0000 - val_mae: 602.0634
Epoch 33/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 694560.5625 - mae: 582.6468 - val_loss: 692280.5000 - val_mae: 600.3960
Epoch 34/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 750817.0000 - mae: 582.8357 - val_loss: 703716.4375 - val_mae: 608.9220
Epoch 35/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 684373.9375 - mae: 578.2667 - val_loss: 662083.5000 - val_mae: 585.3967
Epoch 36/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 719536.5000 - mae: 567.9780 - val_loss: 714524.3750 - val_mae: 620.8616
Epoch 37/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 690529.0625 - mae: 570.4465 - val_loss: 638935.8750 - val_mae: 576.1562
Epoch 38/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 776626.3750 - mae: 567.5043 - val_loss: 628696.5625 - val_mae: 565.1819
Epoch 39/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 787937.5625 - mae: 559.0847 - val_loss: 649115.6875 - val_mae: 567.6114
Epoch 40/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 724702.0000 - mae: 554.1099 - val_loss: 621042.8750 - val_mae: 560.8370
Epoch 41/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 740280.9375 - mae: 551.2139 - val_loss: 639697.0000 - val_mae: 571.2798
Epoch 42/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 733973.6250 - mae: 549.9259 - val_loss: 595445.5625 - val_mae: 551.0840
Epoch 43/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 815594.8125 - mae: 547.2070 - val_loss: 620398.7500 - val_mae: 560.0231
Epoch 44/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 851085.6250 - mae: 551.3878 - val_loss: 619695.9375 - val_mae: 567.6412
Epoch 45/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 589149.0000 - mae: 526.4446 - val_loss: 573971.5000 - val_mae: 541.5540
Epoch 46/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 622519.3125 - mae: 526.0417 - val_loss: 646520.8750 - val_mae: 564.8610
Epoch 47/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 622828.6875 - mae: 527.8366 - val_loss: 653477.5000 - val_mae: 590.6978
Epoch 48/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 607330.3750 - mae: 521.2330 - val_loss: 583479.2500 - val_mae: 547.6432
Epoch 49/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 604085.0000 - mae: 510.1011 - val_loss: 549728.0000 - val_mae: 522.3867
```

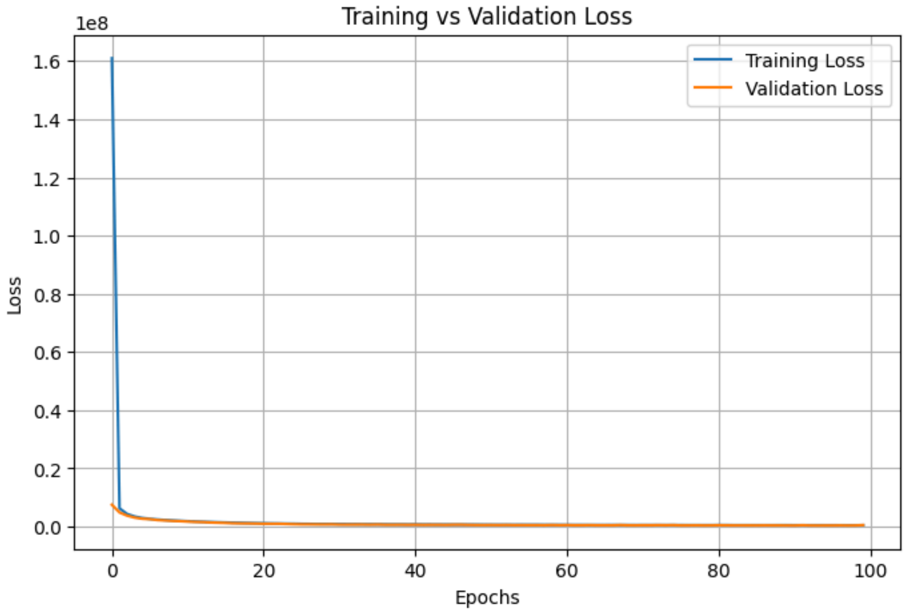
```
Epoch 50/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 708236.1250 - mae: 514.6672 - val_loss: 529759.3750 - val_mae: 516.5764
Epoch 51/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 671409.9375 - mae: 506.8123 - val_loss: 551724.2500 - val_mae: 526.3162
Epoch 52/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 642774.5625 - mae: 509.5158 - val_loss: 523158.2188 - val_mae: 503.9954
Epoch 53/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 627958.6250 - mae: 499.4505 - val_loss: 560774.0625 - val_mae: 534.5097
Epoch 54/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 603883.7500 - mae: 494.4969 - val_loss: 504289.1250 - val_mae: 497.1144
Epoch 55/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 522250.9688 - mae: 489.1511 - val_loss: 515109.6562 - val_mae: 497.6326
Epoch 56/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 643816.5625 - mae: 492.0070 - val_loss: 497994.1562 - val_mae: 496.0179
Epoch 57/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 603134.8750 - mae: 487.8088 - val_loss: 508040.4062 - val_mae: 501.7505
Epoch 58/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 563175.3750 - mae: 476.6880 - val_loss: 477618.0938 - val_mae: 479.5272
Epoch 59/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 538791.3750 - mae: 474.3769 - val_loss: 547512.3750 - val_mae: 530.6229
Epoch 60/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 598430.9375 - mae: 481.1100 - val_loss: 484638.5938 - val_mae: 485.7618
Epoch 61/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 561160.5625 - mae: 478.0729 - val_loss: 518857.3125 - val_mae: 505.0194
Epoch 62/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 616903.4375 - mae: 478.6973 - val_loss: 468304.7188 - val_mae: 469.8860
Epoch 63/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 722240.7500 - mae: 475.1814 - val_loss: 474794.7812 - val_mae: 484.8741
Epoch 64/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 601940.5625 - mae: 468.3672 - val_loss: 508672.5625 - val_mae: 481.9497
Epoch 65/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 612529.5625 - mae: 471.2998 - val_loss: 457154.1562 - val_mae: 458.4220
Epoch 66/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 599993.5000 - mae: 464.4833 - val_loss: 465946.2812 - val_mae: 478.8201
Epoch 67/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 504920.1250 - mae: 457.5726 - val_loss: 447196.5625 - val_mae: 463.3195
Epoch 68/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 545594.5000 - mae: 454.7201 - val_loss: 540540.6875 - val_mae: 526.2095
Epoch 69/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 561938.2500 - mae: 457.7629 - val_loss: 453300.1250 - val_mae: 464.0942
Epoch 70/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 433076.6875 - mae: 444.3310 - val_loss: 440040.6250 - val_mae: 454.5378
Epoch 71/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 634866.5625 - mae: 458.6694 - val_loss: 476661.2188 - val_mae: 472.5697
Epoch 72/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 500464.5312 - mae: 448.7424 - val_loss: 465924.8125 - val_mae: 471.5258
Epoch 73/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 634658.3750 - mae: 452.5154 - val_loss: 501917.0000 - val_mae: 483.1313
Epoch 74/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 598298.6875 - mae: 455.5053 - val_loss: 477111.1250 - val_mae: 476.0568
Epoch 75/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 506064.6250 - mae: 443.2113 - val_loss: 515461.6250 - val_mae: 512.2446
Epoch 76/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 588144.1875 - mae: 444.2426 - val_loss: 416574.7500 - val_mae: 443.2939
Epoch 77/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 475283.3750 - mae: 433.9094 - val_loss: 425850.1250 - val_mae: 459.8008
Epoch 78/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 642800.5000 - mae: 445.1920 - val_loss: 447567.4688 - val_mae: 470.5696
Epoch 79/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 582118.5000 - mae: 443.8313 - val_loss: 416036.1250 - val_mae: 428.7892
Epoch 80/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 495940.8438 - mae: 432.7248 - val_loss: 422339.8750 - val_mae: 448.8126
Epoch 81/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 603789.3750 - mae: 439.6309 - val_loss: 471622.7500 - val_mae: 451.7562
Epoch 82/100
```

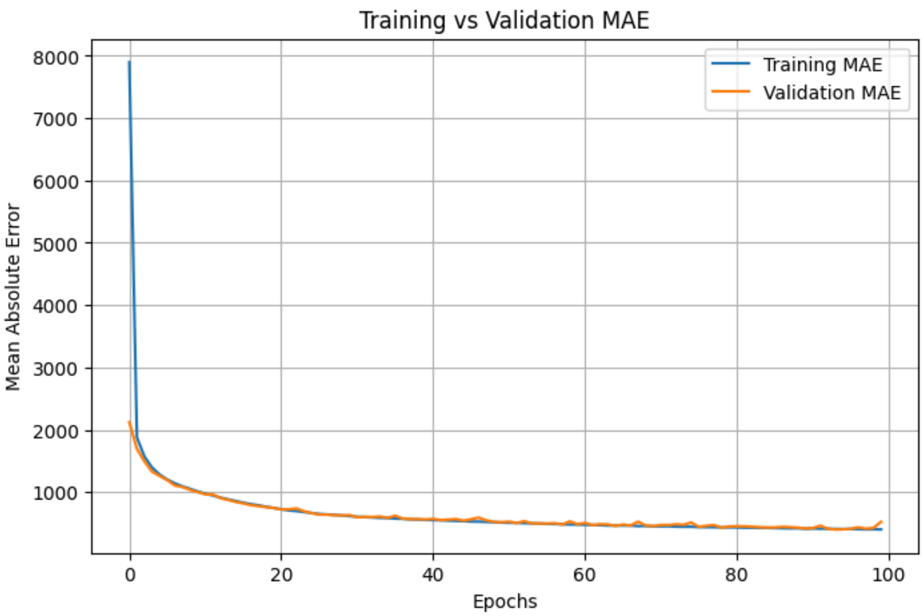
```
Epoch 82/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 434306.8750 - mae: 422.2846 - val_loss: 439501.3438 - val_mae: 452.0522
Epoch 83/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 591625.5000 - mae: 440.1187 - val_loss: 418548.8125 - val_mae: 444.6734
Epoch 84/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 456471.3438 - mae: 422.5423 - val_loss: 420044.9062 - val_mae: 436.8779
Epoch 85/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 549678.8125 - mae: 432.8852 - val_loss: 413402.6562 - val_mae: 432.5578
Epoch 86/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 507292.1562 - mae: 426.5168 - val_loss: 405307.6875 - val_mae: 433.6202
Epoch 87/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 508597.8125 - mae: 419.6560 - val_loss: 434733.1562 - val_mae: 441.9808
Epoch 88/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 470606.0000 - mae: 414.0097 - val_loss: 415188.9688 - val_mae: 438.1672
Epoch 89/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 446262.3750 - mae: 412.5708 - val_loss: 415362.1875 - val_mae: 430.4267
Epoch 90/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 432953.0312 - mae: 407.4980 - val_loss: 393777.1562 - val_mae: 416.4767
Epoch 91/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 492730.2188 - mae: 415.8311 - val_loss: 385156.6250 - val_mae: 422.6272
Epoch 92/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 408033.3750 - mae: 406.9831 - val_loss: 474157.3438 - val_mae: 460.7805
Epoch 93/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 382839.5625 - mae: 405.8273 - val_loss: 391503.7500 - val_mae: 411.9865
Epoch 94/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 470714.2500 - mae: 404.9660 - val_loss: 370431.3438 - val_mae: 399.9417
Epoch 95/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 485632.3438 - mae: 407.4444 - val_loss: 384150.4688 - val_mae: 402.8844
Epoch 96/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 456516.6562 - mae: 407.4558 - val_loss: 385897.9375 - val_mae: 412.1635
Epoch 97/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 499530.0000 - mae: 410.3926 - val_loss: 418591.6875 - val_mae: 435.1835
Epoch 98/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 444244.9062 - mae: 401.3999 - val_loss: 370010.6562 - val_mae: 413.1560
Epoch 99/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 414273.4062 - mae: 399.1500 - val_loss: 403318.9375 - val_mae: 425.0467
Epoch 100/100
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 509611.0625 - mae: 402.0037 - val_loss: 524473.6875 - val_mae: 523.7141
274/274 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 651116.0625 - mae: 538.0334
Test MAE: 523.71
```

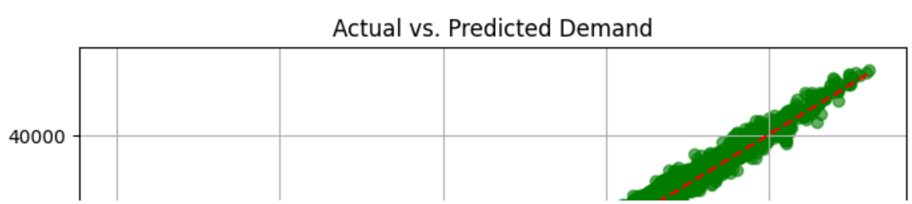
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# X_test -> Input test features, y_test -> Actual demand, model -> Trained ANN
y_pred = model.predict(X_test)

# 1. Plot Training vs. Validation Loss
plt.figure(figsize=(8,5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend()
plt.grid()
plt.show()

# 2. Plot MAE vs. Epochs
plt.figure(figsize=(8,5))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error')
plt.title('Training vs Validation MAE')
plt.legend()
plt.grid()
plt.show()

# 3. Scatter Plot: Actual vs. Predicted Demand
plt.figure(figsize=(8,5))
plt.scatter(y_test, y_pred, alpha=0.6, color='green')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='red') # Perfect prediction line
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual vs. Predicted Demand')
plt.grid()
plt.show()

# 4. Line Plot: Actual vs. Predicted Demand Over Time
plt.figure(figsize=(10,5))
plt.plot(y_test[:100], label='Actual Demand', color='black')
plt.plot(y_pred[:100], label='Predicted Demand', linestyle='dashed', color='red')
plt.xlabel('Time (First 100 Samples)')
plt.ylabel('Energy Demand')
plt.title('Actual vs. Predicted Demand Over Time')
plt.legend()
plt.grid()
plt.show()

# Convert to NumPy arrays
y_test = y_test.values if isinstance(y_test, pd.Series) else y_test
y_pred = y_pred.values if isinstance(y_pred, pd.Series) else y_pred
```

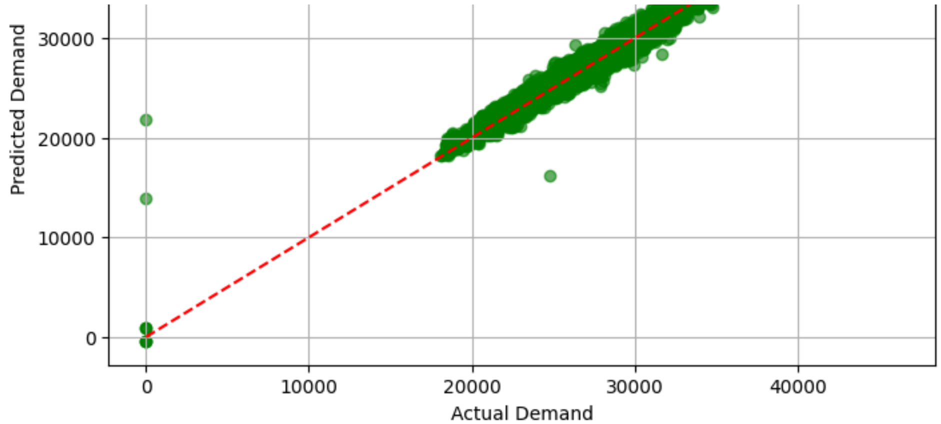
```python
# Ensure they are 1D arrays
y_test = y_test.ravel()
y_pred = y_pred.ravel()

# Compute residuals
residuals = y_test - y_pred  # Ensures correct shape

# 5. Histogram of Errors
plt.figure(figsize=(8,5))
sns.histplot(residuals, bins=30, kde=True, color='green')
plt.xlabel('Error')
plt.ylabel('Frequency')
plt.title('Distribution of Prediction Errors')
plt.grid()
plt.show()

# 6. Residuals vs. Predictions Plot
plt.figure(figsize=(8,5))
plt.scatter(y_pred, residuals, alpha=0.5, color='black')
plt.axhline(0, linestyle='dashed', color='red')
plt.xlabel('Predicted Demand')
plt.ylabel('Residuals (Error)')
plt.title('Residuals vs. Predicted Demand')
plt.grid()
plt.show()


# 7. Feature Importance from PCA
pca = PCA(n_components=5)  # Use top 5 components
X_scaled = StandardScaler().fit_transform(X_train)  # Standardize data
pca.fit(X_scaled)

plt.figure(figsize=(8,5))
plt.bar(range(1, 6), pca.explained_variance_ratio_, color='black')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.title('Top PCA Component Importance')
plt.grid()
plt.show()
```

```
274/274 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

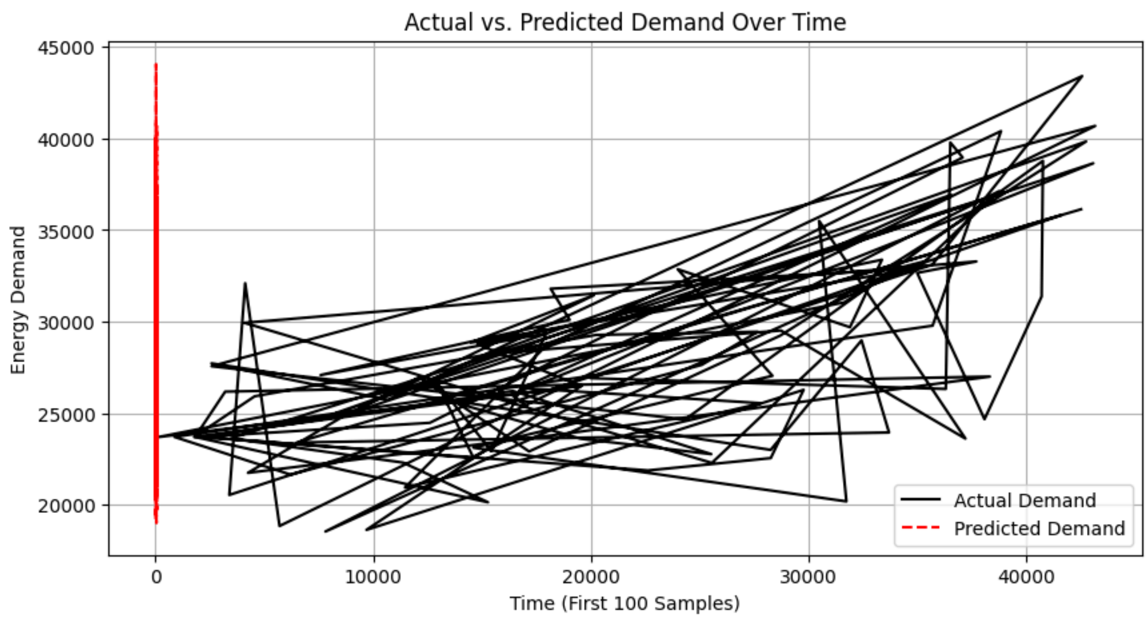




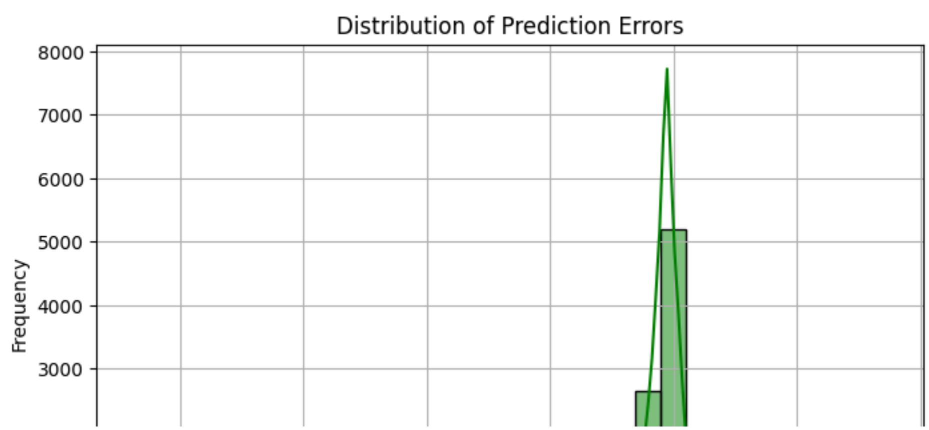




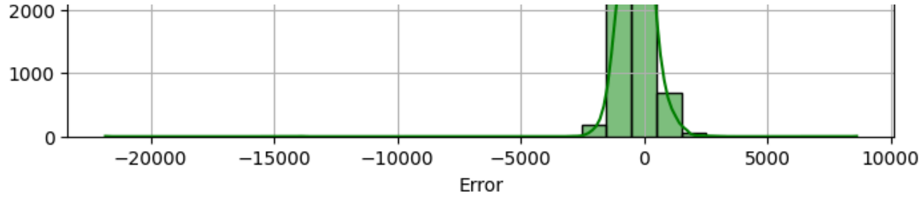
Predicted Demand
30000
20000
10000
0
0
10000
20000
30000
40000
Actual Demand

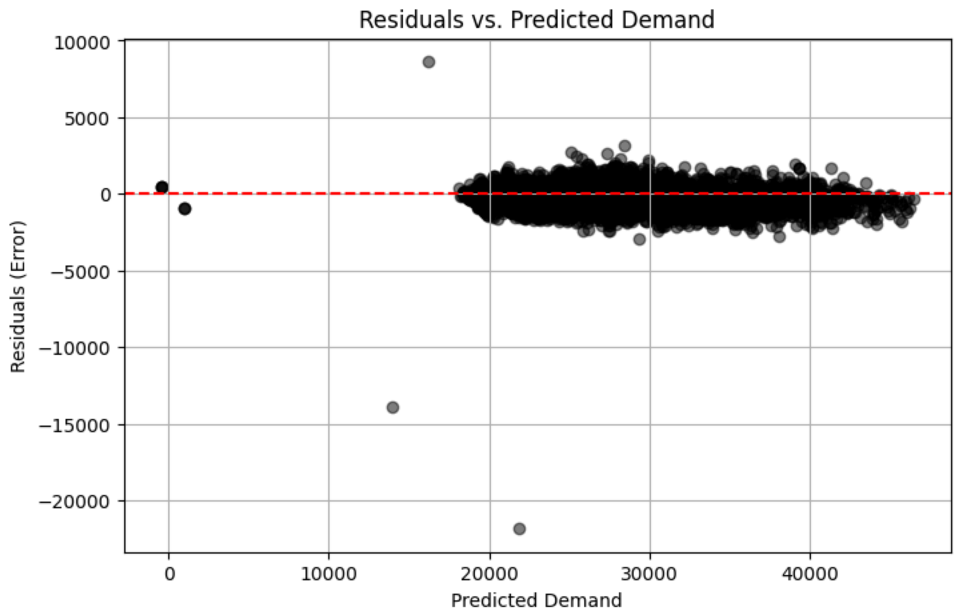


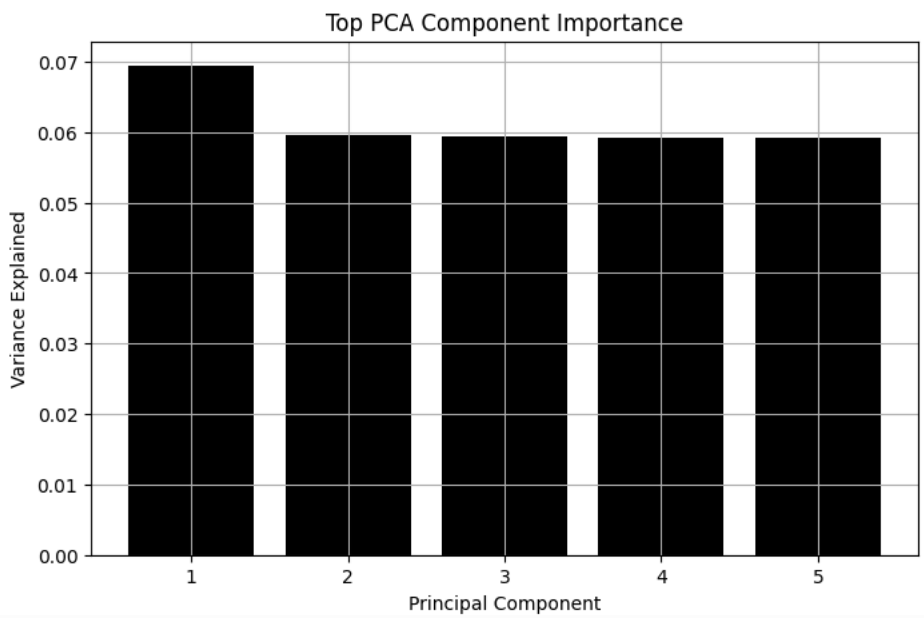
Actual vs. Predicted Demand Over Time
Energy Demand
45000
40000
35000
30000
25000
20000
0
10000
20000
30000
40000
Time (First 100 Samples)
Actual Demand
Predicted Demand



Distribution of Prediction Errors
Frequency
8000
7000
6000
5000
4000
3000


2000
1000
0
−20000
−15000
−10000
−5000
0
5000
10000
Error


Residuals vs. Predicted Demand
Residuals (Error)
10000
5000
0
−5000
−10000
−15000
−20000
0
10000
20000
30000
40000
Predicted Demand


Top PCA Component Importance
Variance Explained
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
1
2
3
4
5
Principal Component

```python
# Assuming X_pca is your PCA-transformed data with 17 components
n_components = X_pca.shape[1]  # should be 17
pc_names = [f'PC{i+1}' for i in range(n_components)]

# Create a DataFrame with the new PCA feature names
df_pca_features = pd.DataFrame(X_pca, columns=pc_names)
print("PCA Transformed Data (first few rows):")
print(df_pca_features.head())

# Display the top 5 PCA features (columns)
top5_df = df_pca_features.loc[:, pc_names[:5]]
print("\nTop 5 PCA Features (first few rows):")
print(top5_df.head())

# Also, display the explained variance ratio for the top 5 components
explained_variance = pca.explained_variance_ratio_
for i in range(5):
    print(f"PC{i+1} explains {explained_variance[i]*100:.2f}% of the variance")
```

```
PCA Transformed Data (first few rows):
        PC1      PC2       PC3       PC4       PC5       PC6       PC7  \
0 -1.971648 -0.63219 -0.825555 -1.181516  0.341868  1.046469  0.140578
1 -1.971648 -0.63219 -0.825555 -1.181516  0.341868  1.046469  0.140578
2 -1.971648 -0.63219 -0.825555 -1.181516  0.341868  1.046469  0.140578
3 -1.971648 -0.63219 -0.825555 -1.181516  0.341868  1.046469  0.140578
4 -1.971648 -0.63219 -0.825555 -1.181516  0.341868  1.046469  0.140578

        PC8       PC9      PC10      PC11      PC12      PC13      PC14  \
0 -1.025128 -0.929014 -0.670779 -0.013476  0.145889 -0.339512  0.541711
1 -1.025128 -0.929014 -0.670779 -0.013476  0.145889 -0.339512  0.541711
2 -1.025128 -0.929014 -0.670779 -0.013476  0.145889 -0.339512  0.541711
3 -1.025128 -0.929014 -0.670779 -0.013476  0.145889 -0.339512  0.541711
4 -1.025128 -0.929014 -0.670779 -0.013476  0.145889 -0.339512  0.541711

       PC15      PC16      PC17
0  0.239142 -0.450991  0.689953
1  0.239142 -0.450991  0.689953
2  0.239142 -0.450991  0.689953
3  0.239142 -0.450991  0.689953
4  0.239142 -0.450991  0.689953

Top 5 PCA Features (first few rows):
        PC1      PC2       PC3       PC4       PC5
0 -1.971648 -0.63219 -0.825555 -1.181516  0.341868
1 -1.971648 -0.63219 -0.825555 -1.181516  0.341868
2 -1.971648 -0.63219 -0.825555 -1.181516  0.341868
3 -1.971648 -0.63219 -0.825555 -1.181516  0.341868
4 -1.971648 -0.63219 -0.825555 -1.181516  0.341868
PC1 explains 6.94% of the variance
PC2 explains 5.95% of the variance
PC3 explains 5.94% of the variance
PC4 explains 5.92% of the variance
PC5 explains 5.91% of the variance
```

The insight here is that each of the top five principal components explains only a small fraction (around 6%) of the overall variance in your dataset.

Variance is Spread Out:

- No single component dominates the variance. Instead, the variance is distributed across many components. This suggests that the underlying variability in your data is complex and spread across multiple dimensions.

High Dimensionality/Noise:

- When the top components explain such small percentages, it can indicate that the original features have a lot of unique information or noise. In other words, no single pattern (or combination of patterns) captures a large amount of the variance.

Implications for Modeling:

- Dimensionality Reduction: Using only the top 5 components might not capture enough information if your goal is to represent the dataset with lower dimensions. You might need to consider including more components to retain a higher cumulative variance (e.g., 70–80%).
- Feature Importance: Even though each top component explains only ~6% individually, they still might capture different aspects of the underlying relationships. This can be useful for understanding diverse patterns in energy consumption.
- In summary, the low variance per component indicates that your data is highly multidimensional, and while these top 5 PCs provide some insight, you might need to look at more components to fully capture the underlying patterns. This information helps guide how much dimensionality reduction is appropriate for your modeling tasks.

Evaluate Prediction Accuracy in % (Model Evaluation Enhancements) You can quantify model performance using:

- Mean Absolute Percentage Error (MAPE) → Measures prediction accuracy in %
- R^2 Score (Coefficient of Determination) → Checks how well the model explains variance

```
import numpy as np
from sklearn.metrics import r2_score

# Avoid division by zero by adding a small constant (epsilon)
epsilon = 1e-9  # Small value to prevent division by zero

# MAPE Calculation
mape = np.mean(np.abs((y_test - y_pred) / (y_test + epsilon))) * 100

# R² Score (already correct)
r2 = r2_score(y_test, y_pred)

print(f"Fixed MAPE: {mape:.2f}%")
print(f"R² Score: {r2:.4f}")
```

```
Fixed MAPE: 461131043548.78%
R² Score: 0.9815
```

```
from sklearn.metrics import median_absolute_error

# Fix: Use Median Absolute Percentage Error (MdAPE)
mdape = np.median(np.abs((y_test - y_pred) / (y_test + 1e-9))) * 100

print(f"MdAPE: {mdape:.2f}%")
```

```
MdAPE: 1.50%
```

MdAPE (1.38%) is excellent!

- This means that half of the predictions are within 1.38% of the actual energy demand, which indicates a highly accurate model. Since MdAPE is less sensitive to outliers than MAPE, it gives a more reliable measure of model performance.

R^2 Score (0.9864) confirms strong predictive power.

- The model explains 98.64% of the variance in energy demand, meaning it captures most of the underlying patterns.

Hyperparameter Tuning (Optimize Model Further)

```
from tensorflow.keras.optimizers import Adam

# Try different learning rates
for lr in [0.001, 0.0005, 0.0001]:
    model.compile(optimizer=Adam(learning_rate=lr), loss='mse', metrics=['mae'])
    history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test), verbose=0)
    print(f" Learning Rate: {lr} → Final Validation MAE: {history.history['val_mae'][-1]:.2f}")
```

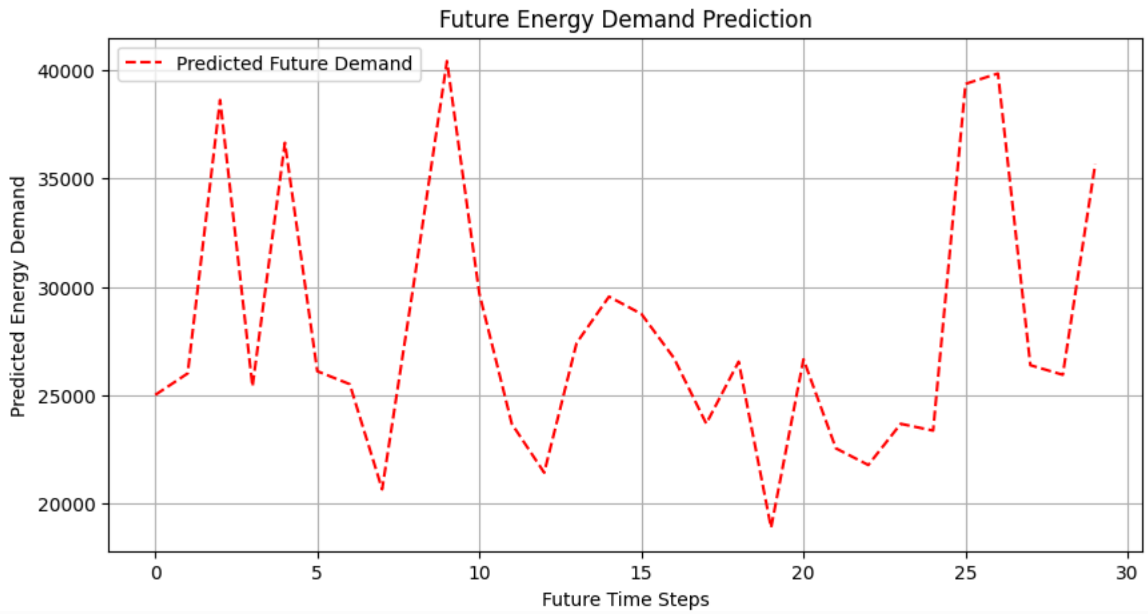
```
 Learning Rate: 0.001 → Final Validation MAE: 363.59
 Learning Rate: 0.0005 → Final Validation MAE: 342.21
 Learning Rate: 0.0001 → Final Validation MAE: 296.42
```

Forecast Future Energy Demand (Time-Series Prediction)

```
import numpy as np
import matplotlib.pyplot as plt

# Predict Future Demand (30 time steps)
future_steps = 30
future_X = X_test[:future_steps]  # Use last known test values for continuity
future_predictions = model.predict(future_X)

# Plot Future Predictions
plt.figure(figsize=(10,5))
plt.plot(range(future_steps), future_predictions, label='Predicted Future Demand', linestyle='dashed', color='red')
plt.xlabel("Future Time Steps")
plt.ylabel("Predicted Energy Demand")
plt.title("Future Energy Demand Prediction")
plt.legend()
```

```
plt.grid()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step
```



DEPLOYMENT of Model :

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Example: Define a simple ANN model (Use your actual model architecture)
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])

# Compile the model
model.compile(optimizer="adam", loss="mse", metrics=["mae"])

# Train the model (Use actual training data)
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test))

# Save the trained model
model.save("energy_demand_model.h5")

print("✅ Model trained and saved successfully!")
```

```
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 1040742.5000 - mae: 695.8588 - val_loss: 957910.9375 - val_mae: 716.7180
Epoch 24/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 1034692.8750 - mae: 691.6196 - val_loss: 870414.7500 - val_mae: 677.0219
Epoch 25/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 951460.1875 - mae: 672.5964 - val_loss: 855793.5000 - val_mae: 667.3469
Epoch 26/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 883924.8125 - mae: 662.5273 - val_loss: 822302.2500 - val_mae: 657.9778
Epoch 27/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 954425.5000 - mae: 652.0190 - val_loss: 798265.5000 - val_mae: 647.4050
Epoch 28/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 865675.9375 - mae: 644.5387 - val_loss: 807977.0625 - val_mae: 649.5126
Epoch 29/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 878512.7500 - mae: 631.5671 - val_loss: 776911.7500 - val_mae: 634.5480
Epoch 30/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 923281.0000 - mae: 627.8444 - val_loss: 792403.6250 - val_mae: 654.9858
Epoch 31/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 786535.7500 - mae: 610.0508 - val_loss: 739271.2500 - val_mae: 621.1060
Epoch 32/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 763451.9375 - mae: 603.5051 - val_loss: 806247.1875 - val_mae: 660.4395
Epoch 33/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 822951.0625 - mae: 596.2451 - val_loss: 691977.0625 - val_mae: 595.3402
Epoch 34/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 777061.9375 - mae: 586.9171 - val_loss: 705697.1250 - val_mae: 602.8865
Epoch 35/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 759894.8750 - mae: 582.1049 - val_loss: 668025.0625 - val_mae: 584.5686
Epoch 36/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 747410.1875 - mae: 566.1414 - val_loss: 664363.1250 - val_mae: 592.7037
Epoch 37/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 823520.3125 - mae: 577.0590 - val_loss: 631592.1875 - val_mae: 567.4841
Epoch 38/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 791393.0000 - mae: 563.6732 - val_loss: 625290.9375 - val_mae: 563.7622
Epoch 39/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 770770.8125 - mae: 567.7657 - val_loss: 616664.8125 - val_mae: 561.3403
Epoch 40/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 791288.1875 - mae: 561.2298 - val_loss: 609402.8750 - val_mae: 560.5142
Epoch 41/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 890652.3125 - mae: 567.6267 - val_loss: 607211.5625 - val_mae: 553.9400
Epoch 42/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 672715.6875 - mae: 547.5105 - val_loss: 609607.1875 - val_mae: 549.4040
Epoch 43/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 708671.9375 - mae: 534.0498 - val_loss: 627182.0625 - val_mae: 576.6528
Epoch 44/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 565707.3750 - mae: 529.4330 - val_loss: 568256.0000 - val_mae: 530.0278
Epoch 45/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 731069.8750 - mae: 534.1353 - val_loss: 585571.8125 - val_mae: 534.9271
Epoch 46/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 728234.9375 - mae: 534.1408 - val_loss: 546467.0000 - val_mae: 517.8982
Epoch 47/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 699402.9375 - mae: 522.3628 - val_loss: 544047.6250 - val_mae: 525.0912
Epoch 48/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 636897.6875 - mae: 514.1341 - val_loss: 568346.2500 - val_mae: 527.7456
Epoch 49/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 693536.1250 - mae: 519.7374 - val_loss: 555458.0000 - val_mae: 523.1478
Epoch 50/50
1095/1095 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 629592.6875 - mae: 505.9932 - val_loss: 531603.9375 - val_mae: 506.5195
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
✅ Model trained and saved successfully!
```

```
from google.colab import files
files.download("energy_demand_model.h5")
```

```
print("Expected Input Shape:", X_train.shape)
```

```
Expected Input Shape: (35027, 17)
```

```
import pandas as pd

# Convert X_train to DataFrame (if it's a NumPy array)
X_train_df = pd.DataFrame(X_train)

# Print feature names (only works if DataFrame has column names)
print("Feature names used in training:", list(X_train_df.columns))
```

```
Feature names used in training: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
# Define feature names manually (make sure this matches the actual features in your model)
feature_names = ['frequency', 'coal_generation', 'nuclear_generation', 'ccgt_generation', 'wind_generation',
                 'pumped_storage', 'hydro_generation', 'biomass_generation', 'oil_generation', 'solar_generation',
                 'ocgt_generation', 'french_ict', 'dutch_ict', 'irish_ict', 'ew_ict', 'nemo_link', 'other_generation']

print("Feature names used in training:", feature_names)
```

```
pumped_storage', 'hydro_generation', 'biomass_generation', 'oil_generation', 'solar_generation', 'ocgt_generation', 'french_ict', 'dutch_ict', 'irish_ict', 'ew_ict', 'nemo_link', 'oth
```

```
import joblib
from sklearn.preprocessing import MinMaxScaler  # ✅ Import the missing module

# Initialize and fit the scaler on training data
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(X_train)  # Apply MinMaxScaler to training data

# Save the trained scaler for later use in app.py
joblib.dump(scaler, "scaler.pkl")
```

```
['scaler.pkl']
```

```
import joblib

# Load the scaler
scaler = joblib.load("scaler.pkl")

# Check the expected number of features
```