In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [36]:
```
# time_series = pd.read_csv("groupedWikipedia_de_15min.txt", header=None, names=["R
time_series = np.genfromtxt("groupedWikipedid_jp_10min.txt")
```

In [37]:
```
import numpy as np
from scipy.spatial.distance import euclidean
from fastdtw import fastdtw

# https://ashukumar27.medium.com/similarity-functions-in-python-aa6dfe721035

def manhattan_distance(point1, point2):
    return np.sum(np.abs(point1 - point2))

def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    if norm_vec1 == 0 or norm_vec2 == 0:
        # Handle the case where one of the vectors has zero magnitude
        similarity = 0.0  # or another appropriate value
    else:
        similarity = dot_product / (norm_vec1 * norm_vec2)
    return similarity

def euclidean_distance(vec1, vec2):
    return euclidean(vec1, vec2)

#DTW
# https://ealizadeh.com/blog/introduction-to-dynamic-time-warping/
# https://www.databricks.com/blog/2019/04/30/understanding-dynamic-time-warping.htm

def fast_dtw(vec1, vec2):
    return fastdtw(vec1, vec2)[0]


#Jaccard is not important here because we dont use categories but integers
```

In [64]:
```
# Configuration

# renge of comaprsion is 86 days the value grouped every 15 minutes
orig_r=12
orig_c=144

#two hour in three day before
wind_r=3
wind_c=12
# distance=manhattan_distance #euclidean_distance, cosine_similarity, fast_dtw, man
```

In [65]:
```python
import numpy as np

def extract_submatrices(matrix, k, m):
    n, d = matrix.shape
    submatrices = []

    for i in range(n-k+1):
        for j in range(d):
            submatrix = np.zeros((k, m))

            if(i==n-k and j>d-m):
              break

            for r in range(k):
                for c in range(m):
                  if(j + c < d):
                    row_idx = (i + r)
                    col_idx = (j + c)
                  elif(i+r<n-1):
                    row_idx = (i + r+1)
                    col_idx = (j + c) % d
                  submatrix[r, c] = matrix[row_idx, col_idx]

            submatrices.append(submatrix.flatten())

    return submatrices

# Example usage
# n = 20
# d = 5
# k = 2
# m = 3

# # Create a random matrix for demonstration purposes
# matrix = np.random.randint(1, 100, (n, d))

# print("Original Matrix:")
# print(matrix)

# submatrices = extract_submatrices(matrix, k, m)
# for i, submatrix in enumerate(submatrices):
#     print(f"Submatrix {i + 1}:")
#     print(submatrix)
#     print()
```

In [66]:
```python
import numpy as np
import sys

max_integer = sys.maxsize


def update_matrix(ground_ts, new_element):
    ground_ts = np.append(ground_ts[:, 1:], new_element)
    return ground_ts
```

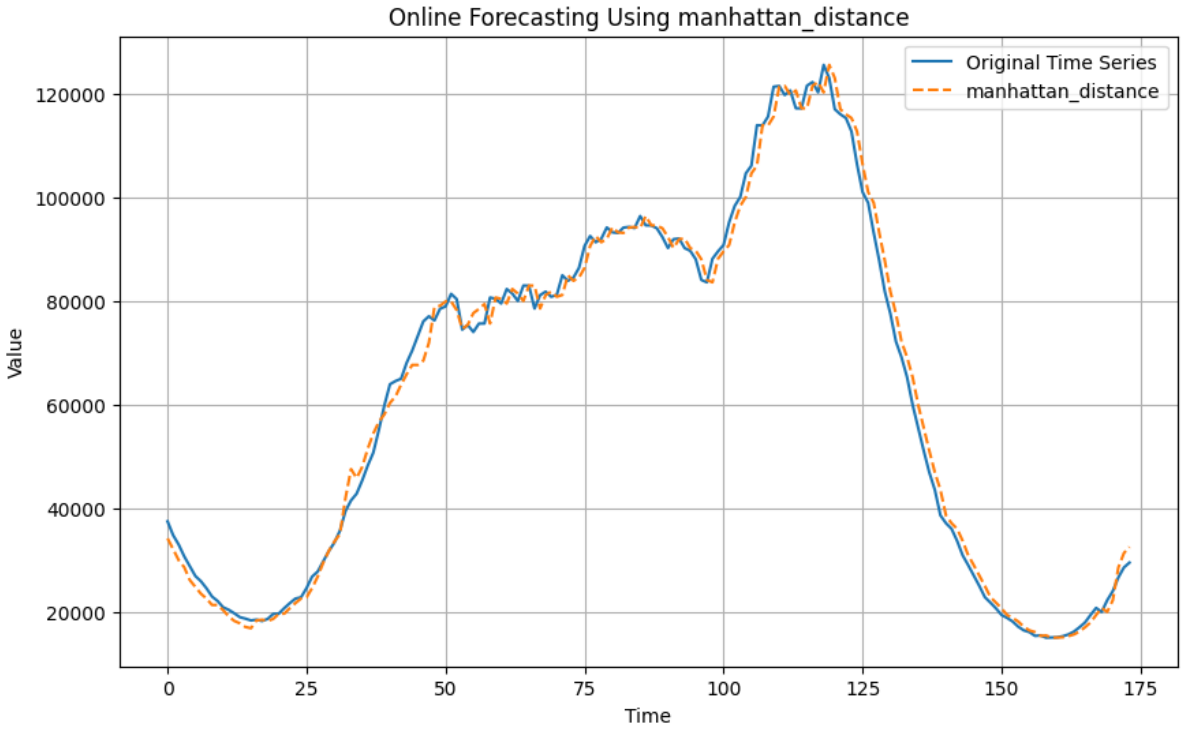
```
def prediction(time_series, orig_r=orig_r, orig_c=orig_c, wind_r=wind_r, wind_c=win

  # Prepare and shape time series
  if(len(time_series)>orig_r*orig_c):
    time_series=time_series[-(orig_r*orig_c):]
  else:
    sparse =np.full((orig_r*orig_c)-len(time_series),max_integer)
    time_series= np.concatenate((sparse, time_series))

  time_series=update_matrix(time_series,0).reshape(orig_r, orig_c)

  # Extract Submatrices
  submatrices = extract_submatrices(time_series, wind_r, wind_c)

  similarity_distances = {
    manhattan_distance: [],
    euclidean_distance: [],
    cosine_similarity:  [],
    fast_dtw: []
  }

  # Calculate the similarity between the main series (e.g the last submatrix) and o
  main_series=submatrices[-1]
  for submatrix in submatrices[:-1]:
     for key in similarity_distances:
      distance = key(main_series[:-1], submatrix[:-1])
      similarity_distances[key].append(distance)

  mst_similar = {
    "manhattan_distance": None,
    "euclidean_distance": None,
    "cosine_similarity":  None,
    "fast_dtw": None
  }

  for key in similarity_distances:
    if(key!=cosine_similarity):
      opt_dist =min(similarity_distances[key])
    else:
      opt_dist =max(similarity_distances[key])
    idx_of_mst_similar=similarity_distances[key].index(opt_dist)
    mst_similar[key.__name__]=submatrices[idx_of_mst_similar][-1]

    #Debug
#   print(similarity_distances[-5:])
#   print("the index is" ,idx_of_mst_similar)
#   print("the main array", main_series)
#   print("the mst simmiler array", mst_similar)
#   print("the forecast is ", mst_similar[-1])
  return mst_similar

# ground_ts= update_matrix(ground_ts, 10)
# ground_ts= update_matrix(ground_ts, 15)
# similarity))
# print(prediction(matrix.flatten()))
```

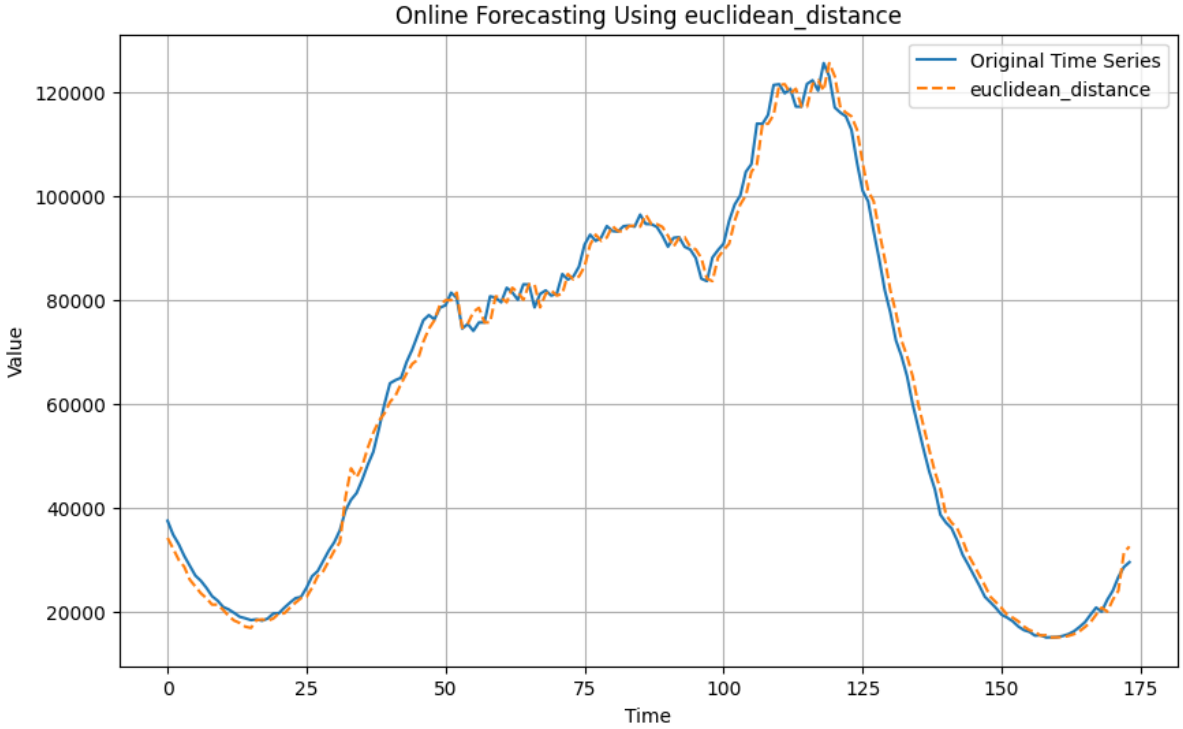
In [67]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

time_steps = len(time_series)

# Split data into training and testing sets
train_size = int(0.9 * time_steps)
train_data = time_series[:train_size]
test_data = time_series[train_size:]

# print(matrix)
# Forecasting on the test data
predictions = {
    "manhattan_distance": [],
    "euclidean_distance": [],
    "cosine_similarity":  [],
    "fast_dtw": []
  }
for t in range(len(train_data), len(time_series)):
    y_pred = prediction(time_series[:t])
    for k in predictions:
      predictions[k].append(y_pred[k])
```

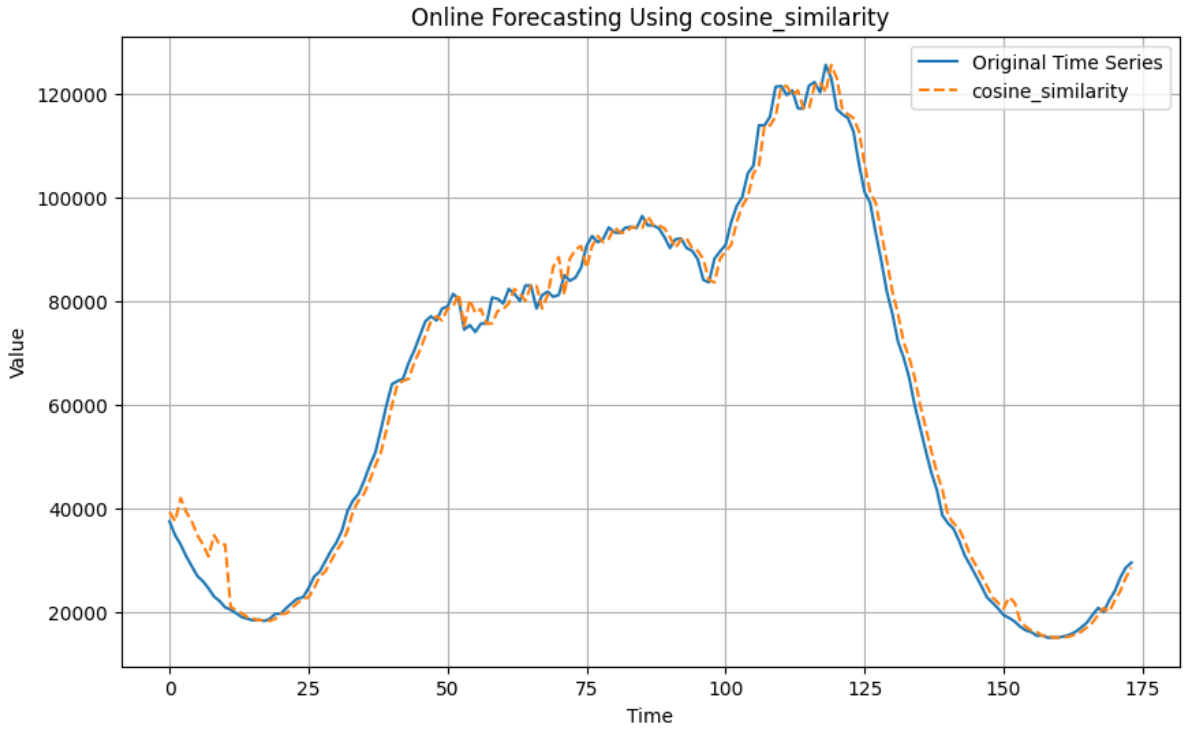
In [68]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
# Calculate the forecasting error (MSE,MAE)
print("========= Mean Squared Error ===========")
for k in predictions:
    mse = mean_squared_error(test_data[:-1], predictions[k][:-1])
    print("MSE "+k+":\n", mse)

print("========= Mean Absolute Error ===========")
for k in predictions:
    mae = mean_absolute_error(test_data[:-1], predictions[k][:-1])
    print("MAE "+k+":\n", mae)

print("========= Mean Absolute Percentage Error ===========")
for k in predictions:
    mape = mean_absolute_percentage_error(test_data[:-1], predictions[k][:-1])
    print("MAPE "+k+":\n", mape)
```

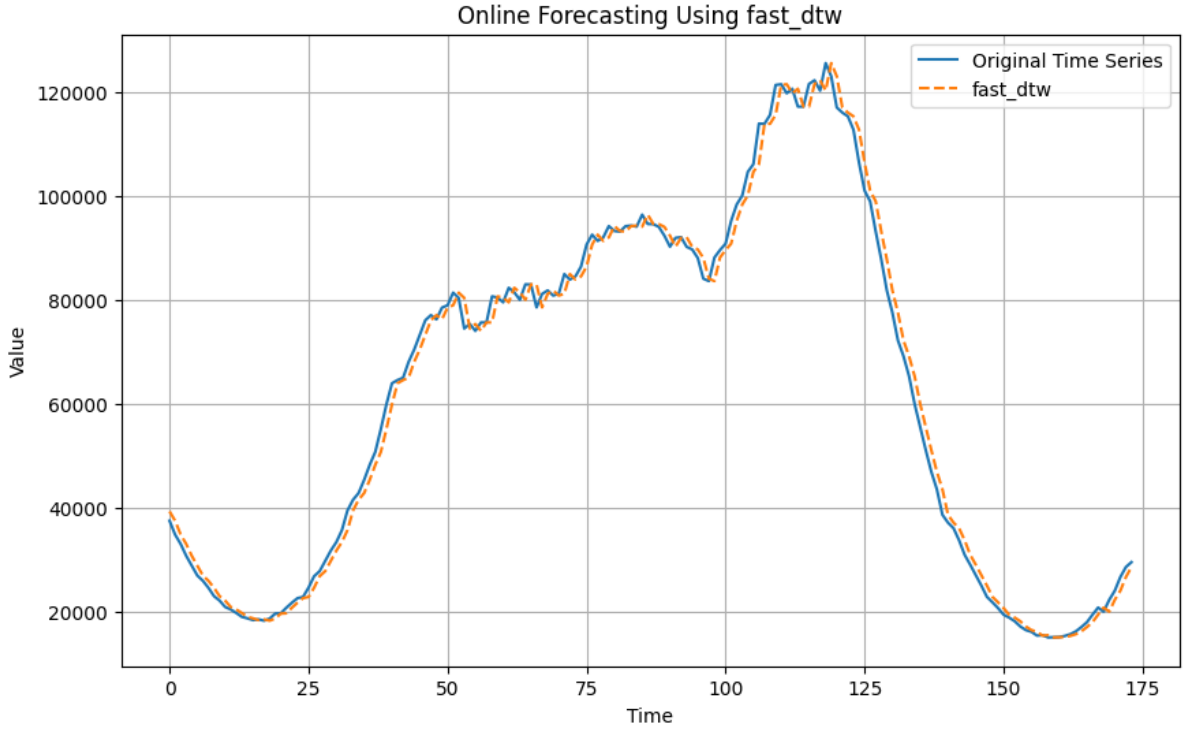
```
========== Mean Squared Error ===========
MSE manhattan_distance:
 7287262.804597701
MSE euclidean_distance:
 6860362.1034482755
MSE cosine_similarity:
 11992495.379310345
MSE fast_dtw:
 6544328.885057472
========== Mean Absolute Error ===========
MAE manhattan_distance:
 2083.057471264368
MAE euclidean_distance:
 2045.057471264368
MAE cosine_similarity:
 2548.735632183908
MAE fast_dtw:
 1966.448275862069
========== Mean Absolute Percentage Error ===========
MAPE manhattan_distance:
 0.04072893429871913
MAPE euclidean_distance:
 0.04102150754423208
MAPE cosine_similarity:
 0.05745897870581711
MAPE fast_dtw:
 0.03789404849610997
```

In [69]:
```python
# Plot original time series and EMA forecast
for k in predictions:
    plt.figure(figsize=(10, 6))
    plt.plot(test_data[:-1], label='Original Time Series')
    plt.plot(predictions[k][:-1], label=k, linestyle='dashed')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title('Online Forecasting Using '+k)
    plt.legend()
    plt.grid(True)
    plt.show()
```









In [48]:
```
# Configuration

# renge of comaprsion is 86 days the value grouped every 15 minutes
orig_r=12
orig_c=144

#two hour in three day before
wind_r=4
```

In [49]:
```
import numpy as np
import sys

max_integer = sys.maxsize


def update_matrix(ground_ts, new_element):
  ground_ts = np.append(ground_ts[1:], new_element)
  return ground_ts

def prediction(time_series, orig_r=orig_r, orig_c=orig_c, wind_r=wind_r, wind_c=win

  # Prepare and shape time series
  if(len(time_series)>orig_r*orig_c):
    time_series=time_series[-(orig_r*orig_c):]
  else:
    sparse =np.full((orig_r*orig_c)-len(time_series),max_integer)
    time_series= np.concatenate((sparse, time_series))

  time_series=update_matrix(time_series,0).reshape(orig_r, orig_c)

  # Extract Submatrices
  submatrices = extract_submatrices(time_series, wind_r, wind_c)

  similarity_distances = {
    manhattan_distance: [],
    euclidean_distance: [],
    cosine_similarity:  [],
    fast_dtw: []
  }

  # Calculate the similarity between the main series (e.g the last submatrix) and o
  main_series=submatrices[-1]
  for submatrix in submatrices[:-1]:
     for key in similarity_distances:
      distance = key(main_series[:-1], submatrix[:-1])
      similarity_distances[key].append(distance)

  mst_similar = {
    "manhattan_distance": None,
    "euclidean_distance": None,
    "cosine_similarity":  None,
    "fast_dtw": None
  }

  for key in similarity_distances:
    if(key!=cosine_similarity):
      opt_dist =min(similarity_distances[key])
    else:
      opt_dist =max(similarity_distances[key])
    idx_of_mst_similar=similarity_distances[key].index(opt_dist)
    mst_similar[key.__name__]=submatrices[idx_of_mst_similar][-1]
```



```
    #Debug
#   print(similarity_distances[-5:])
```

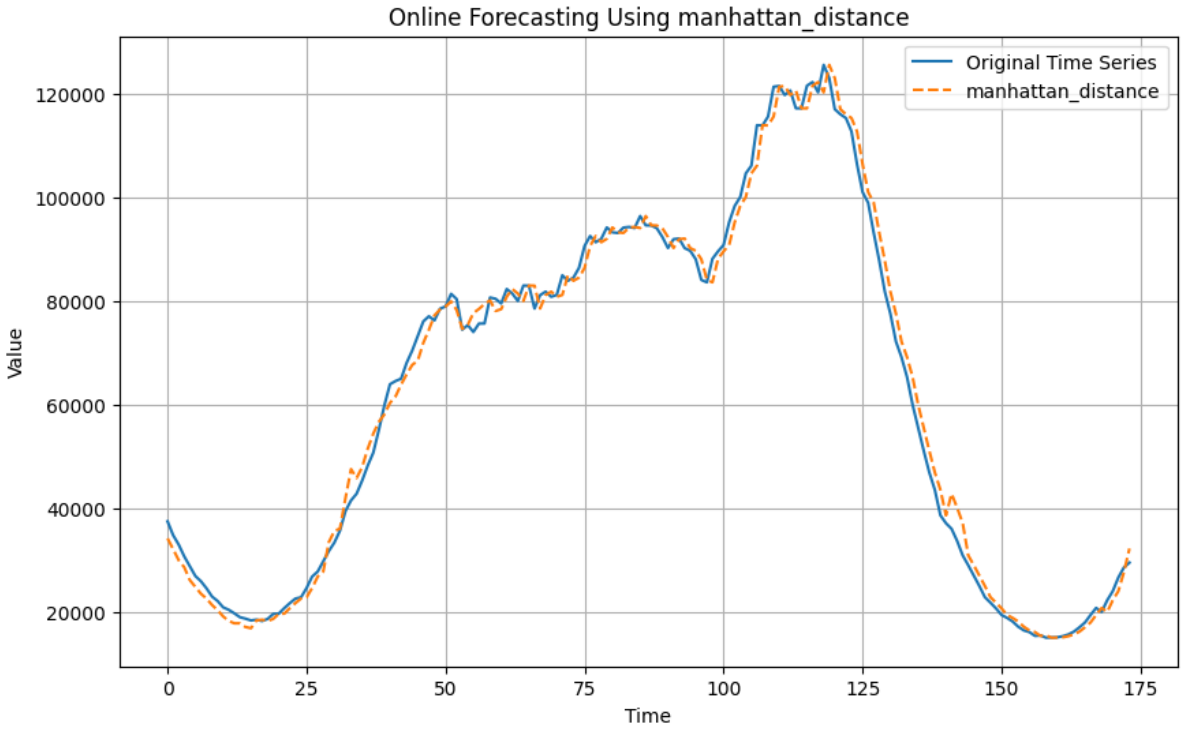
```
#   print("the index is" ,idx_of_mst_similar)
#   print("the main array", main_series)
#   print("the mst simmiler array", mst_similar)
#   print("the forecast is ", mst_similar[-1])
  return mst_similar

# ground_ts= update_matrix(ground_ts, 10)
# ground_ts= update_matrix(ground_ts, 15)
# print(prediction(ground_ts,cosine_similarity))
# print(prediction(matrix.flatten()))
```

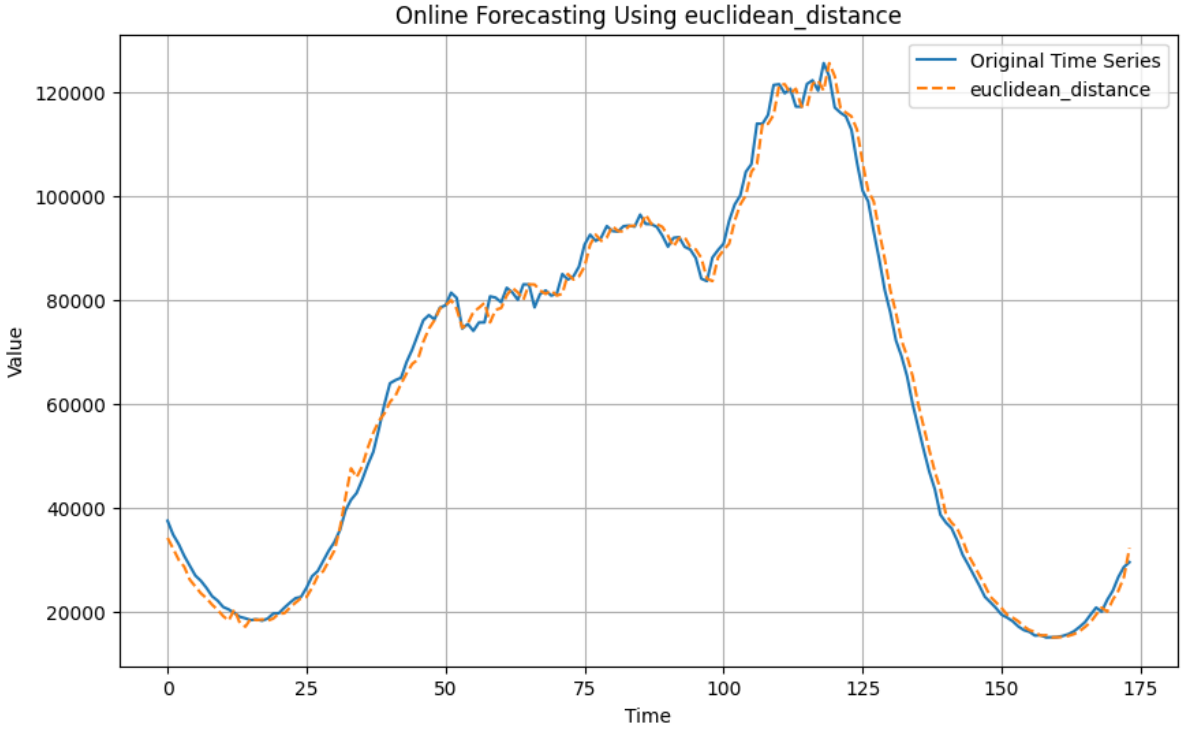
In [50]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

time_steps = len(time_series)

# Split data into training and testing sets
train_size = int(0.9 * time_steps)
train_data = time_series[:train_size]
test_data = time_series[train_size:]

# print(matrix)
# Forecasting on the test data
predictions = {
    "manhattan_distance": [],
    "euclidean_distance": [],
    "cosine_similarity":  [],
    "fast_dtw": []
  }
for t in range(len(train_data), len(time_series)):
    y_pred = prediction(time_series[:t])
    for k in predictions:
      predictions[k].append(y_pred[k])
```

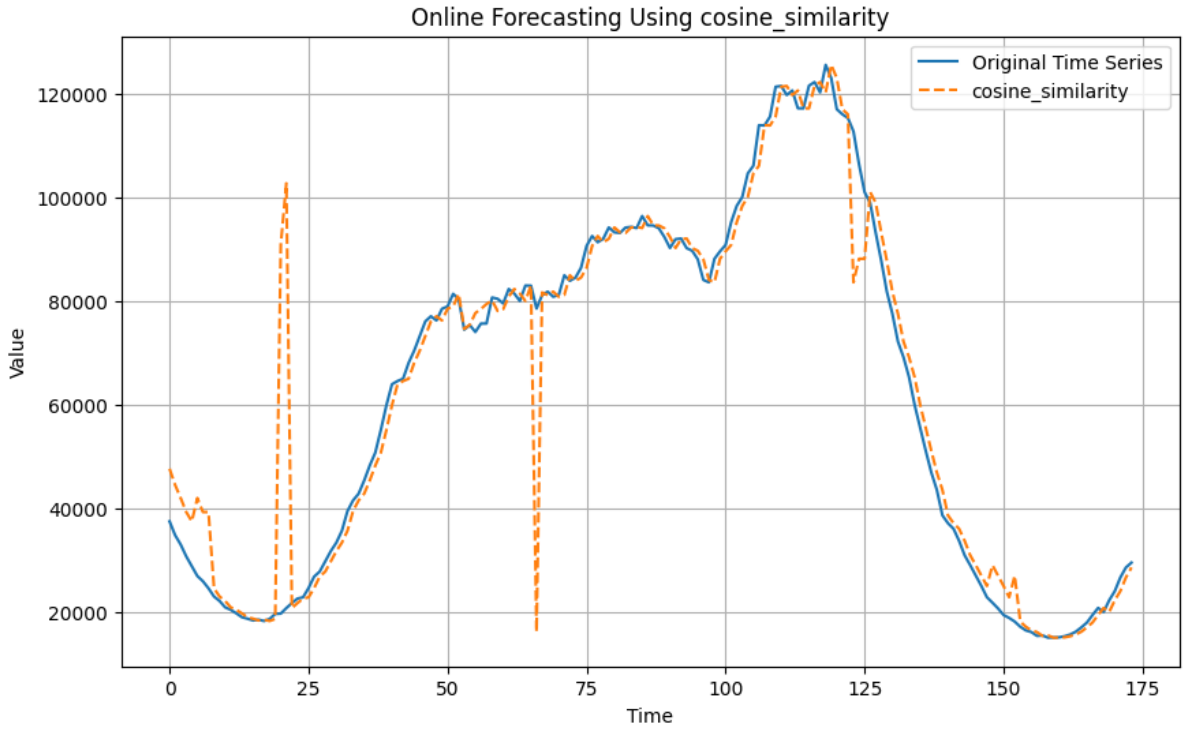
In [55]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
# Calculate the forecasting error (MSE,MAE)
print("========= Mean Squared Error ===========")
for k in predictions:
    mse = mean_squared_error(test_data[:-1], predictions[k][:-1])
    print("MSE "+k+":\n", mse)

print("========= Mean Absolute Error ===========")
for k in predictions:
    mae = mean_absolute_error(test_data[:-1], predictions[k][:-1])
    print("MAE "+k+":\n", mae)

print("========= Mean Absolute Percentage Error ===========")
for k in predictions:
    mape = mean_absolute_percentage_error(test_data[:-1], predictions[k][:-1])
```

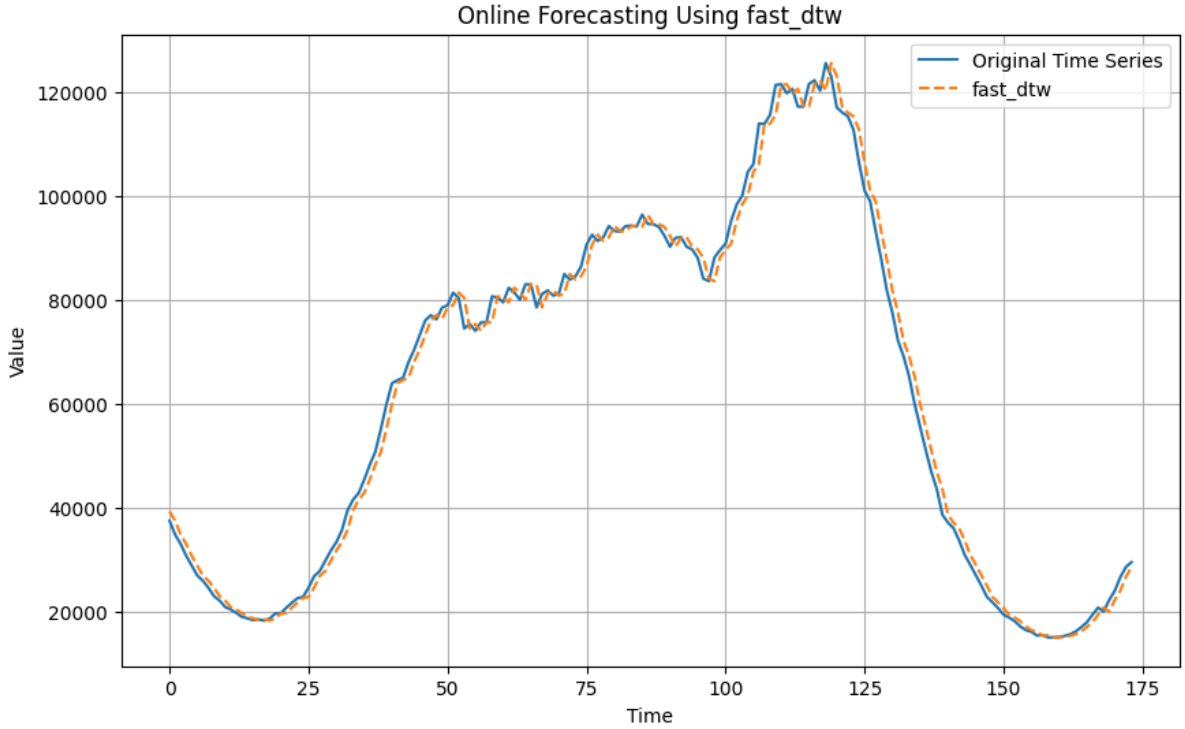
```
========== Mean Squared Error ===========
MSE manhattan_distance:
 7446472.890804598
MSE euclidean_distance:
 6883601.879310345
MSE cosine_similarity:
 110858960.8045977
MSE fast_dtw:
 6544328.885057472
========== Mean Absolute Error ===========
MAE manhattan_distance:
 2124.8563218390805
MAE euclidean_distance:
 2048.729885057471
MAE cosine_similarity:
 3995.9885057471265
MAE fast_dtw:
 1966.448275862069
========== Mean Absolute Percentage Error ===========
MAPE manhattan_distance:
 0.04364063620054577
MAPE euclidean_distance:
 0.04093259765186499
MAPE cosine_similarity:
 0.11038979666837802
MAPE fast_dtw:
 0.03789404849610997
```

In [57]:
```python
# Plot original time series and EMA forecast
for k in predictions:
    plt.figure(figsize=(10, 6))
    plt.plot(test_data[:-1], label='Original Time Series')
    plt.plot(predictions[k][:-1], label=k, linestyle='dashed')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title('Online Forecasting Using '+k)
    plt.legend()
    plt.grid(True)
    plt.show()
```









In [ ]:
```
test_perf = prediction(time_series[:len(train_data)+20])
```