In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
# time_series = pd.read_csv("groupedWikipedia_de_15min.txt", header=None, names=["R
time_series = np.genfromtxt("groupedWikipedia_de_10min.txt")
```

In [6]:
```
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)
    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 [24]:
```
# 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 [25]:
```
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 [26]:

```
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 [27]:

```
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])
```

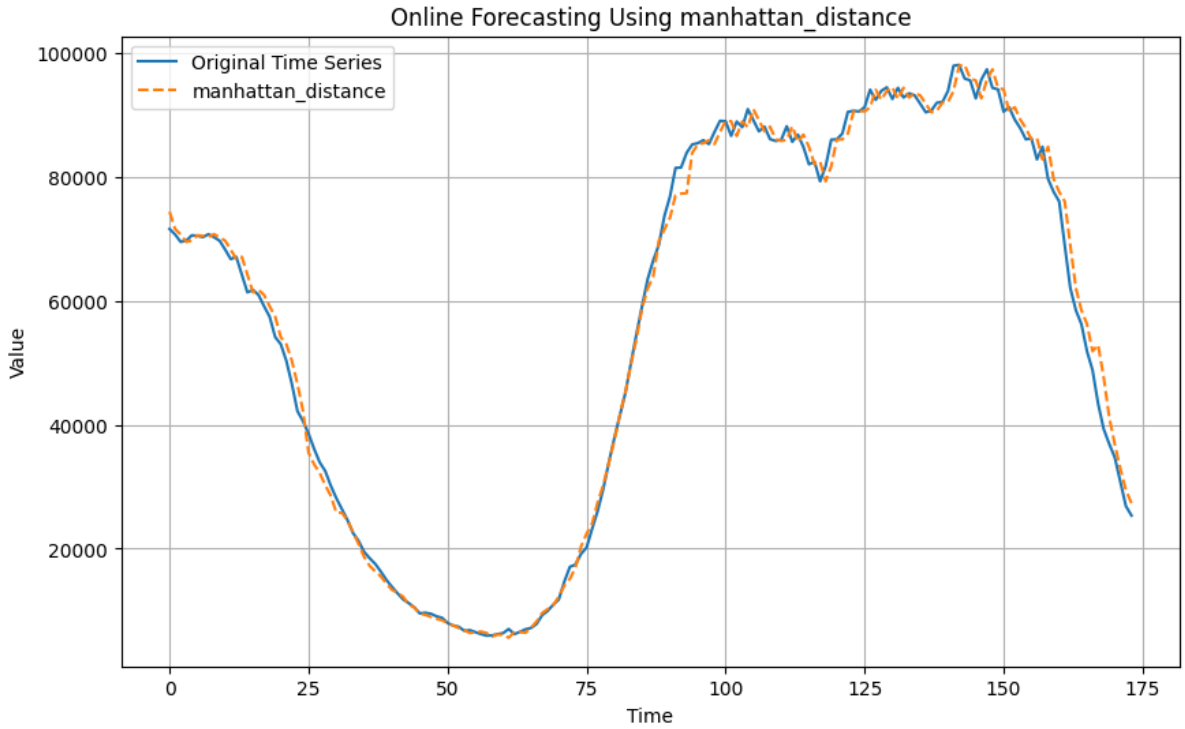
In [28]:

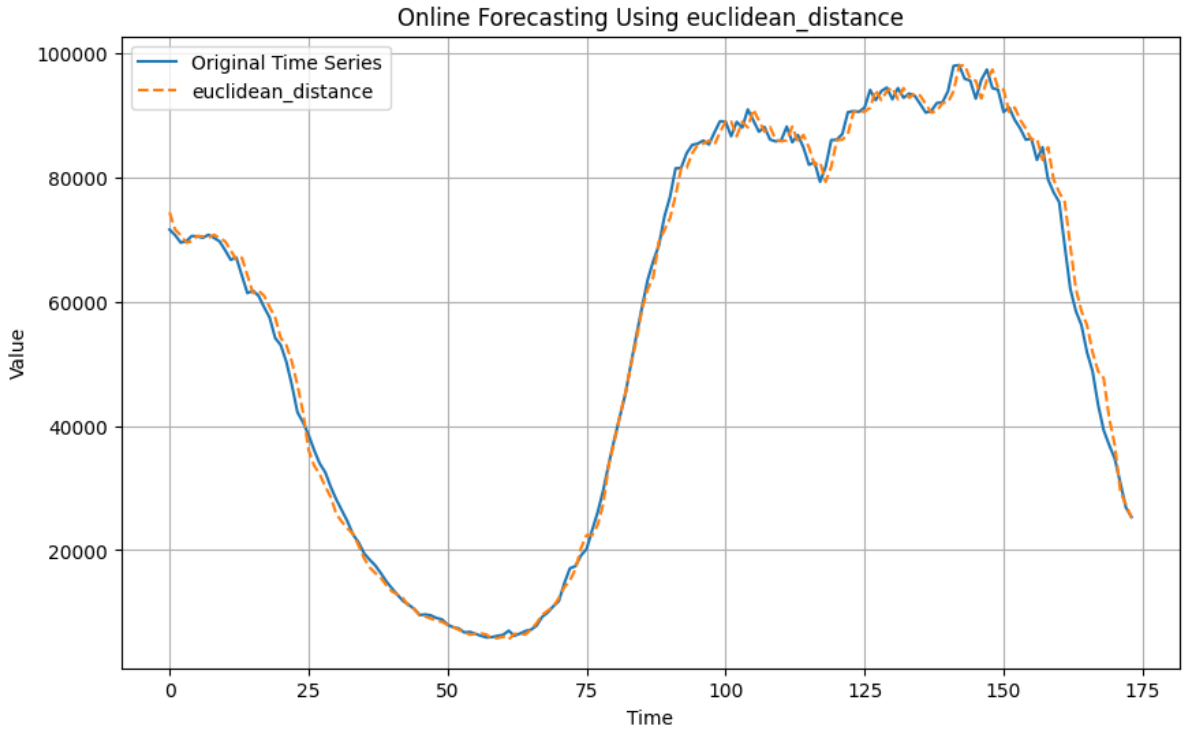
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
# Calculate the forecasting error (MSE,MAE)
print("========== Mean Squared Error ===========")
for k in predictions:
    mse = mean_squared_error(test_data[:-1],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

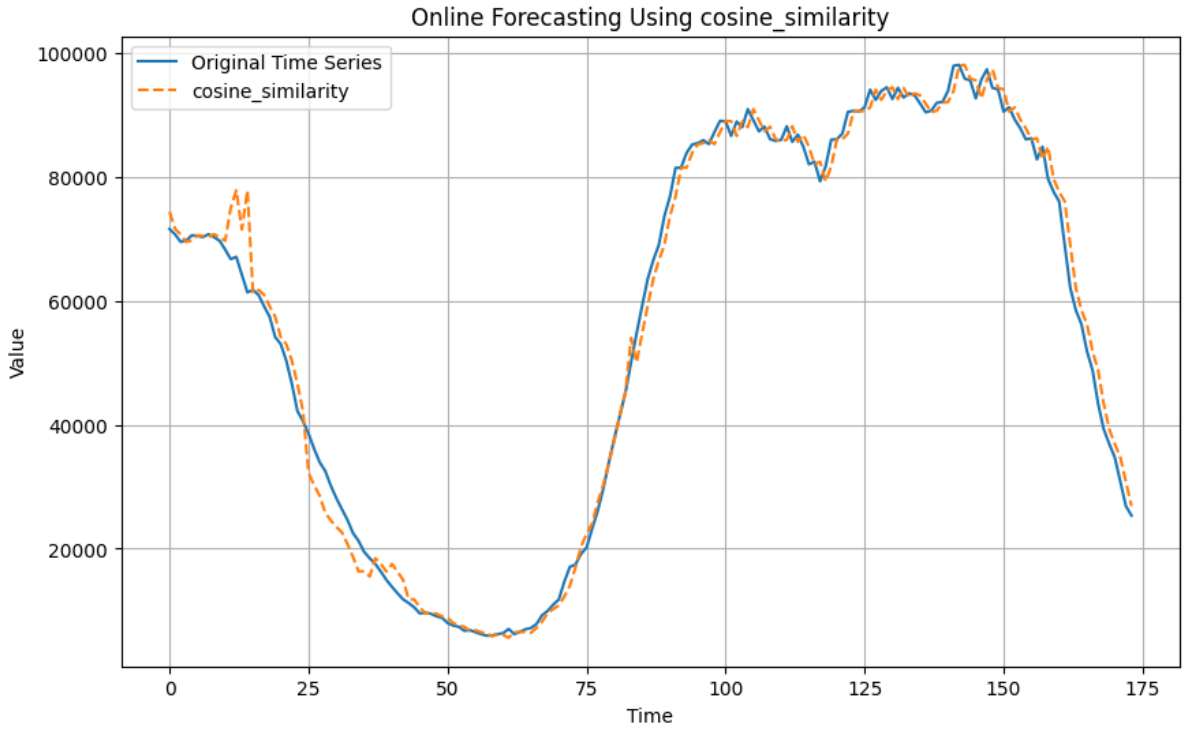
```
========== Mean Squared Error ===========
MSE manhattan_distance:
 4989010.770114942
MSE euclidean_distance:
 4295433.885057472
MSE cosine_similarity:
 9195078.011494253
MSE fast_dtw:
 5310752.471264368
========== Mean Absolute Error ===========
MAE manhattan_distance:
 1555.5977011494253
MAE euclidean_distance:
 1482.1264367816093
MAE cosine_similarity:
 2118.5057471264367
MAE fast_dtw:
 1794.8850574712644
========== Mean Absolute Percentage Error ===========
MAPE manhattan_distance:
 0.03407575466329876
MAPE euclidean_distance:
 0.03312451502226686
MAPE cosine_similarity:
 0.055622465233267575
MAPE fast_dtw:
 0.04523653952610967
```

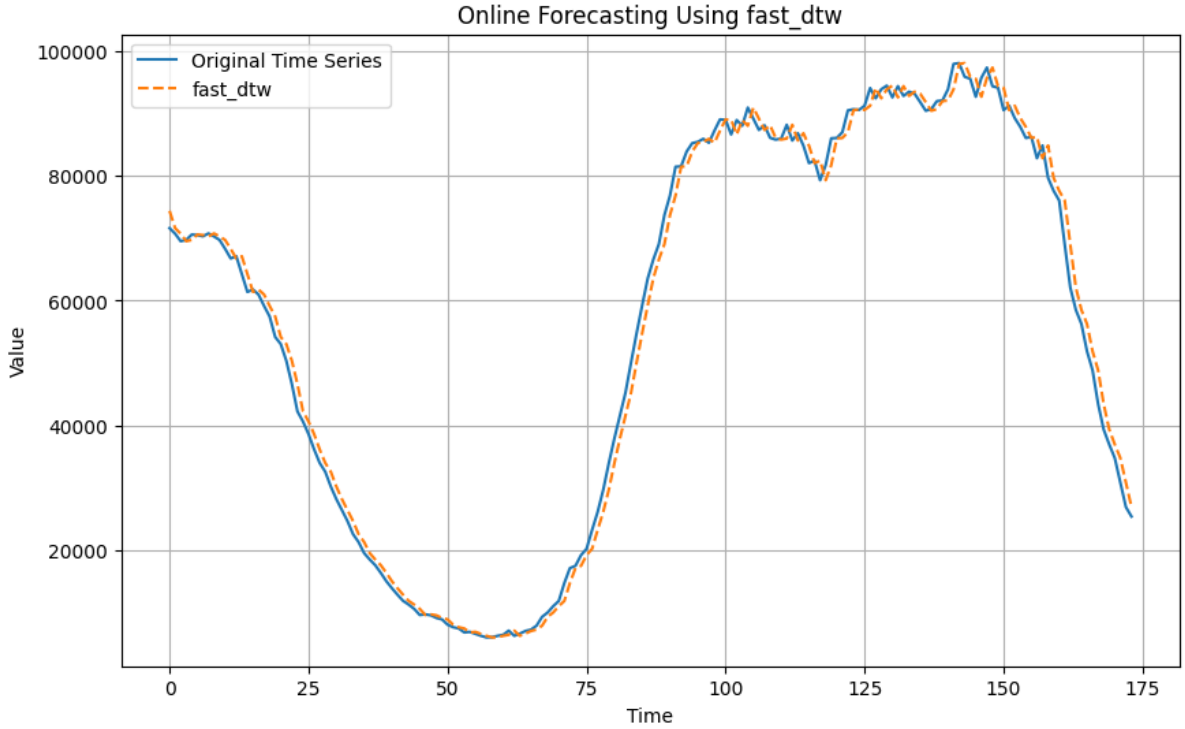
In [29]:

```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()
```


Online Forecasting Using manhattan_distance
Original Time Series
manhattan_distance
Value
Time
100000
80000
60000
40000
20000
0
25
50
75
100
125
150
175


Online Forecasting Using euclidean_distance
Original Time Series
euclidean_distance
Value
Time
100000
80000
60000
40000
20000
0
25
50
75
100
125
150
175





In [17]:
```
# 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
wind_c=9
```

In [18]:
```
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()))
```

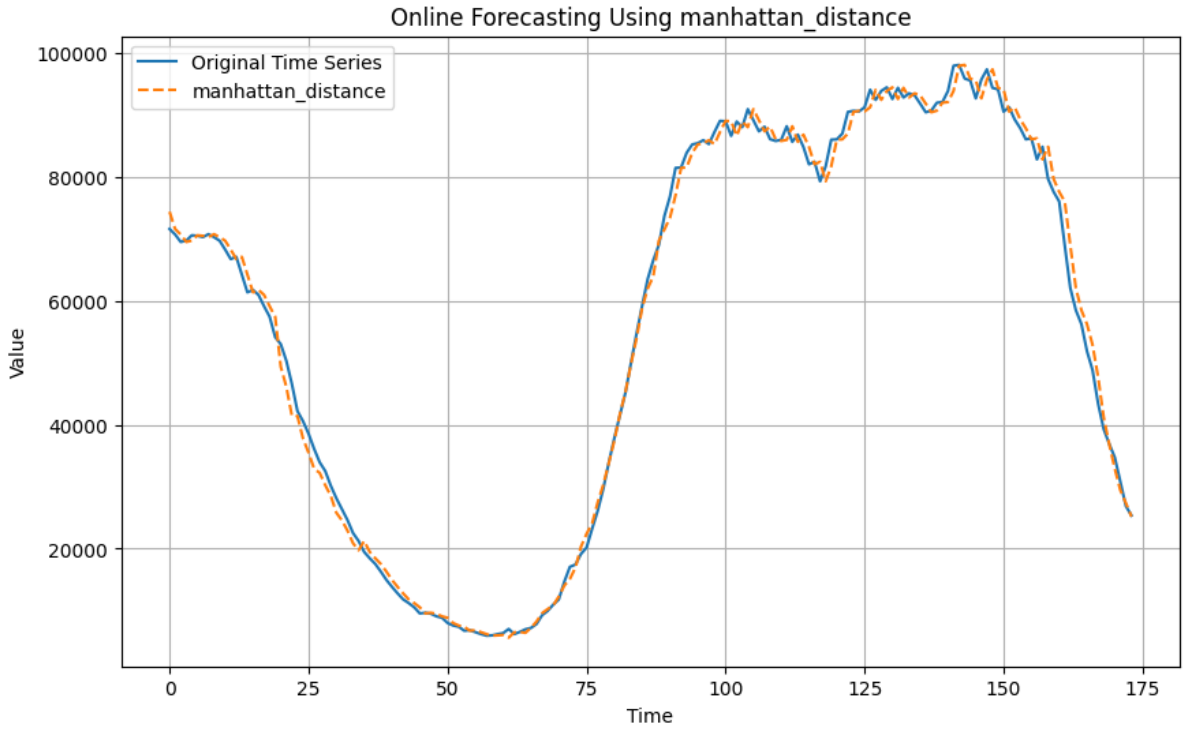
In [19]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

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])
```

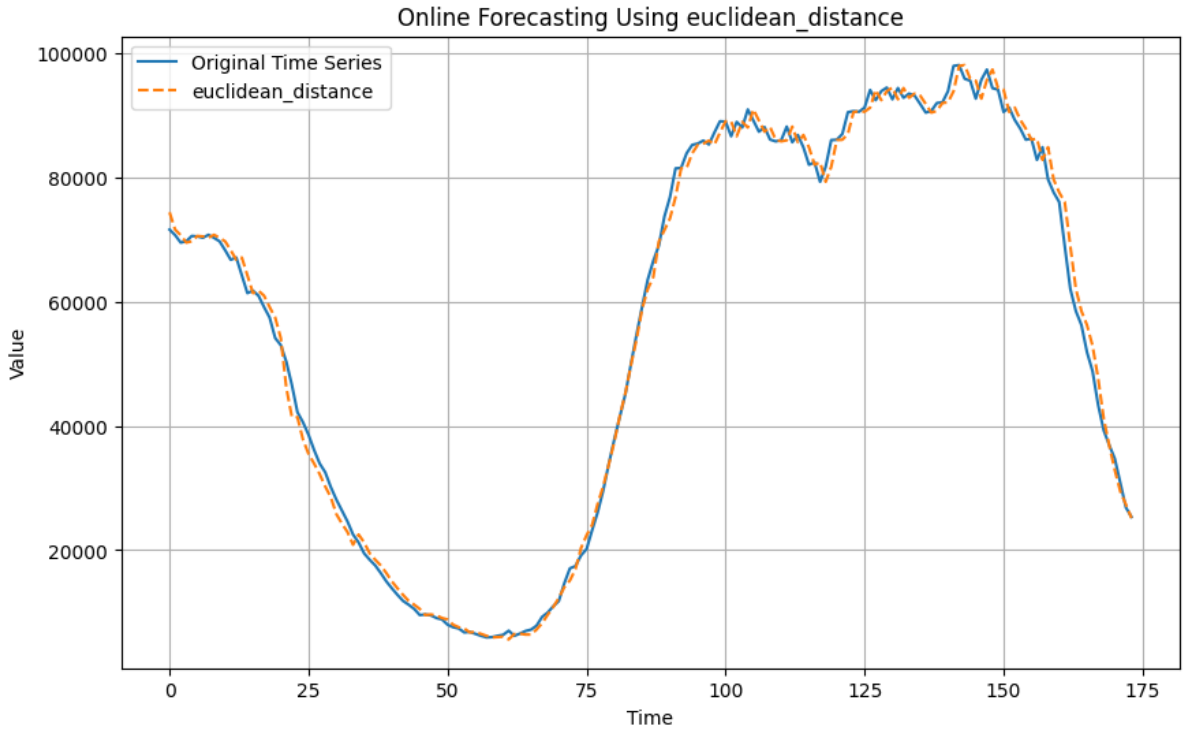
In [22]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
# Calculate the forecasting error (MSE,MAE)
print("========= Mean Squared Error ===========")
for k in predictions:
    mse = mean_squared_error(test_data[:-1], 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)
```

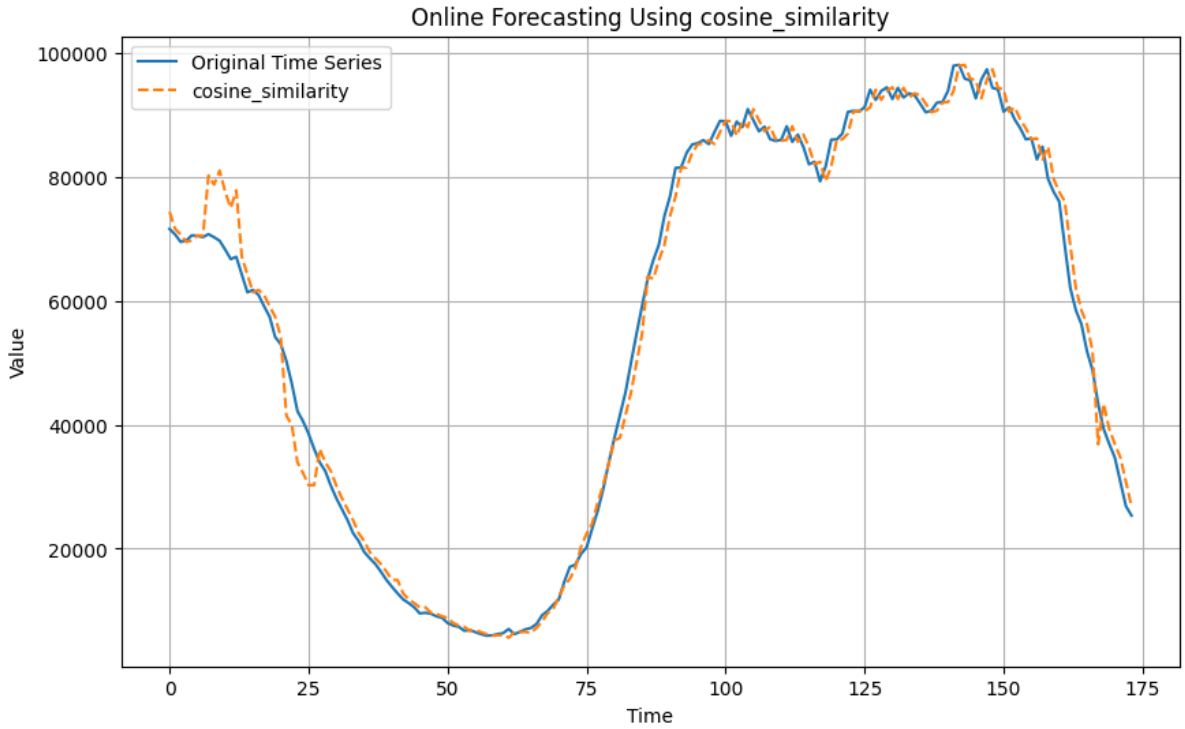
```
========== Mean Squared Error ===========
MSE manhattan_distance:
 3993039.436781609
MSE euclidean_distance:
 3916468.5344827585
MSE cosine_similarity:
 9890553.316091955
MSE fast_dtw:
 5297828.5
========== Mean Absolute Error ===========
MAE manhattan_distance:
 1484.0344827586207
MAE euclidean_distance:
 1475.3045977011495
MAE cosine_similarity:
 2163.9597701149423
MAE fast_dtw:
 1787.2241379310344
========== Mean Absolute Percentage Error ===========
MAPE manhattan_distance:
 0.03525969054669092
MAPE euclidean_distance:
 0.03608434357458899
MAPE cosine_similarity:
 0.05054676943439518
MAPE fast_dtw:
 0.04493440513818453
```

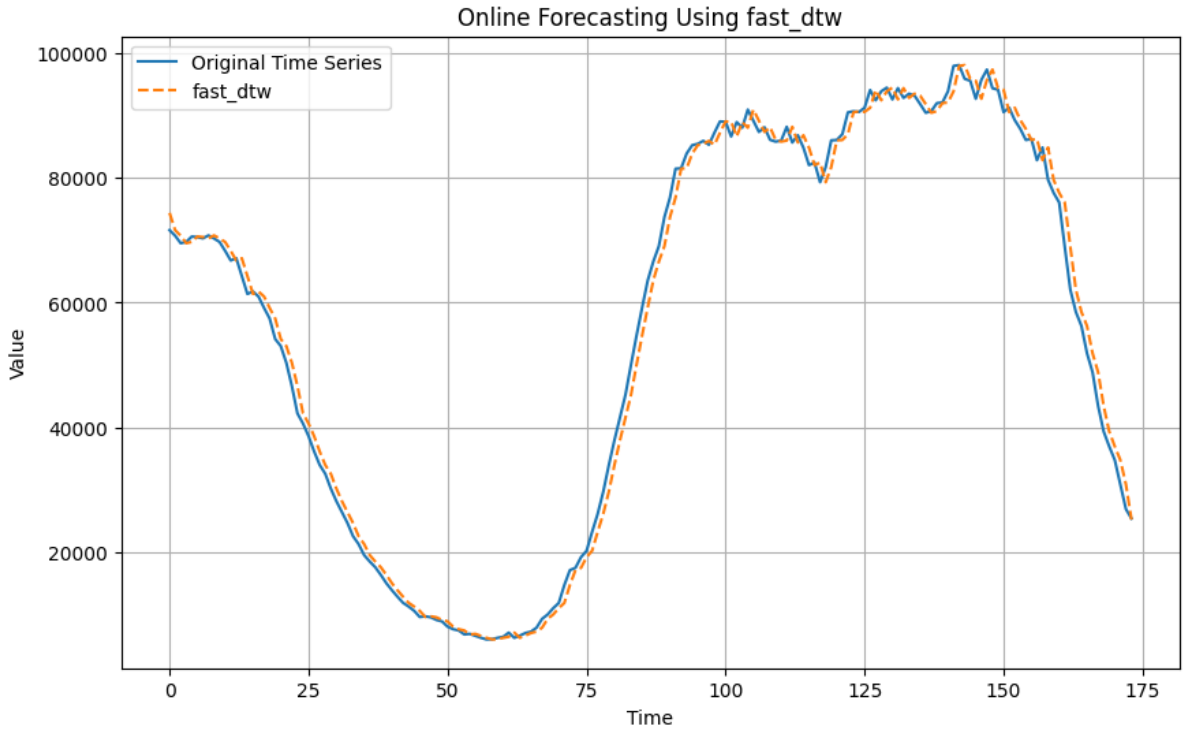
In [23]:

```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()
```


Online Forecasting Using manhattan_distance
Original Time Series
manhattan_distance
Value
Time
100000
80000
60000
40000
20000
0
25
50
75
100
125
150
175


Online Forecasting Using euclidean_distance
Original Time Series
euclidean_distance
Value
Time
100000
80000
60000
40000
20000
0
25
50
75
100
125
150
175





Random Array for Algorithm show

In [14]:
```
import numpy as np

# Create an array of zeros with 7 rows and 96 columns
arr = np.zeros((7, 24))

# Set the value of each element in the array to a random number between 0 and 100
for i in range(7):
```

```
    for j in range(24):
        if i ==6 and j ==23:
            arr[i][j]=0
        else:
            arr[i][j] = np.random.randint(0, 100)

# Print the resulting array
print(arr)
```

```
[[40. 51. 46. 33. 27.  0. 46. 96. 36. 72.  4. 91. 22. 58. 64. 84. 61. 49.
  11. 53. 74.  2. 98. 39.]
 [ 7. 51. 74. 90. 10. 50. 50. 81. 97. 65. 70. 23. 42. 77. 90. 95. 22. 23.
  60. 62. 64. 80. 54. 85.]
 [37. 58. 92. 96. 96.  8. 20. 65. 74. 65. 76. 58. 54. 23. 33. 43. 73.  9.
  13. 27. 40.  0. 35. 49.]
 [37. 95. 92. 73. 76. 82. 78. 85. 64. 19. 63. 50. 90. 46. 53.  9. 24.  3.
   8. 21. 60. 27. 22. 41.]
 [29. 78. 41. 83. 49. 84. 22. 42. 66. 71. 63. 37. 43. 55. 91. 17. 36. 66.
  17. 37. 40. 16. 94. 15.]
 [71. 31. 13. 17. 50. 61. 86. 11. 83. 49. 56. 15. 83. 81. 21. 66. 99. 42.
  69. 24. 69. 62. 22. 69.]
 [69. 85. 61.  1. 23.  5. 88. 16. 39. 71. 21. 37. 31. 90. 74.  4. 93. 14.
  50. 62. 34. 12. 34.  0.]]
```

In [16]:

```
import pandas as pd
from tabulate import tabulate

# Create a DataFrame with labeled rows and columns
df = pd.DataFrame(arr, index=['-6d','-5d','-4d','-3d','-2d','-1d', 'current Day'])

# Print the resulting DataFrame
ndf=df.loc[:,15:]

print(tabulate(ndf, headers='keys', tablefmt='psql'))
```

```
+-------------+------+------+------+------+------+------+------+------+------+
|             |   15 |   16 |   17 |   18 |   19 |   20 |   21 |   22 |   23 |
|-------------+------+------+------+------+------+------+------+------+------|
| -6d         |   84 |   61 |   49 |   11 |   53 |   74 |    2 |   98 |   39 |
| -5d         |   95 |   22 |   23 |   60 |   62 |   64 |   80 |   54 |   85 |
| -4d         |   43 |   73 |    9 |   13 |   27 |   40 |    0 |   35 |   49 |
| -3d         |    9 |   24 |    3 |    8 |   21 |   60 |   27 |   22 |   41 |
| -2d         |   17 |   36 |   66 |   17 |   37 |   40 |   16 |   94 |   15 |
| -1d         |   66 |   99 |   42 |   69 |   24 |   69 |   62 |   22 |   69 |
| current Day |    4 |   93 |   14 |   50 |   62 |   34 |   12 |   34 |    0 |
+-------------+------+------+------+------+------+------+------+------+------+
```

In [5]:

```
!pip install tabulate
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting tabulate
  Downloading tabulate-0.9.0-py3-none-any.whl (35 kB)
Installing collected packages: tabulate
  WARNING: The script tabulate is installed in '/home/mohamadbashar.disoki/.local/
bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warnin
g, use --no-warn-script-location.
Successfully installed tabulate-0.9.0
```

In [49]:
```
ndf=df.iloc[0:2,15:18]
print(ndf)
t=np.reshape(ndf.values,-1).reshape(6,1)

print(t)
```

```
       15    16    17
-6d  84.0  61.0  49.0
-5d  95.0  22.0  23.0
[[84.]
 [61.]
 [49.]
 [95.]
 [22.]
 [23.]]
```

In [ ]:

In [ ]: