In [10]:

```python
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

# Example data: coordinates and labels (0: different route, 1: same
data = np.array([
    [17.2404, 78.4293, 1],  # Shamshabad, Hyderabad
    [17.5453, 78.3408, 1],  # Narkuda, Hyderabad
    [17.1582, 78.1874, 0],  # Shabad, Hyderabad
])

# Split data into features (X) and labels (y)
X = data[:, :2]
y = data[:, 2]

# Normalize/Scale features
X_scaled = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

# Train KNN model
k = 2  # Number of neighbors
knn_model = KNeighborsClassifier(n_neighbors=k)
knn_model.fit(X_scaled, y)
```

Out[10]:

```
KNeighborsClassifier(n_neighbors=2)
```

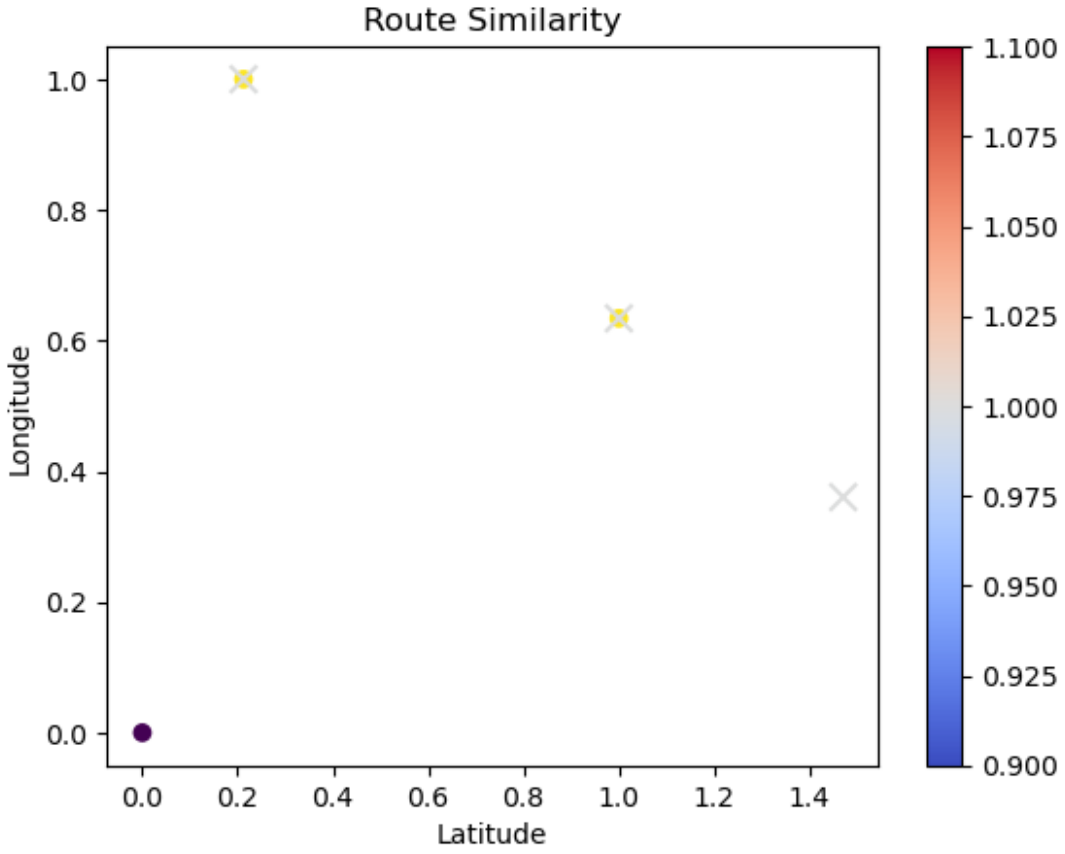
In [11]:

```python
# Custom input latitude and longitude points
input_points = np.array([
    [17.2404, 78.4293],  # Shamshabad, Hyderabad
    [17.5453, 78.3408],  # Narkuda, Hyderabad
    [17.7275, 78.2746],  # Rajiv Gandhi International Airport (not
])
```

In [12]:

```
# Normalize/Scale input points
input_scaled = (input_points - X.min(axis=0)) / (X.max(axis=0) - X.

# Predict route similarity
predictions = knn_model.predict(input_scaled)

# Visualization
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, cmap='viridis')
plt.scatter(input_scaled[:, 0], input_scaled[:, 1], c=predictions,
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Route Similarity')
plt.colorbar()
plt.show()

print("Predictions:", predictions)
```

```
C:\Users\kaila\anaconda3\lib\site-packages\sklearn\neighbors\_classifi
cation.py:228: FutureWarning: Unlike other reduction functions (e.g. `
skew`, `kurtosis`), the default behavior of `mode` typically preserves
the axis it acts along. In SciPy 1.11.0, this behavior will change: th
e default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no
longer be accepted. Set `keepdims` to True or False to avoid this warn
ing.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

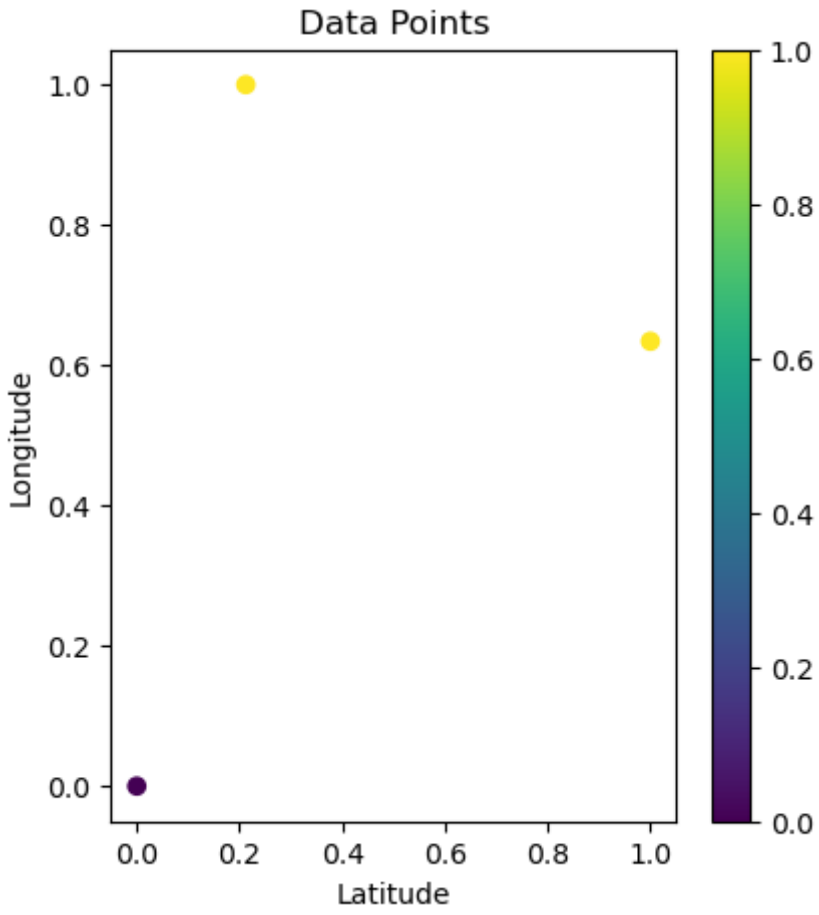



```
Predictions: [1. 1. 1.]
```

In [13]:

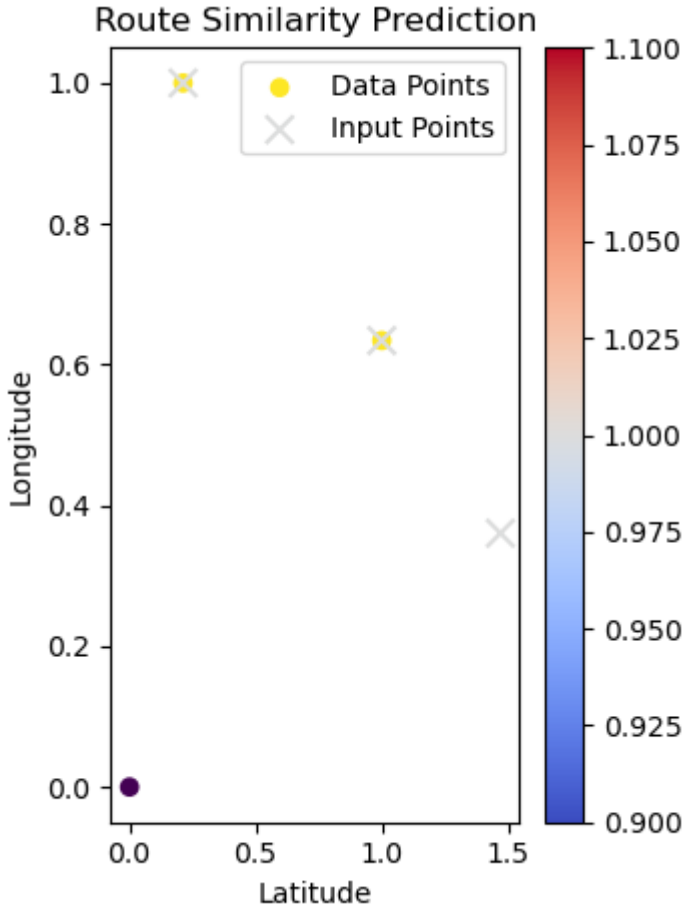
```
# Visualization: Scatter plot of data points
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, cmap='viridis')
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Data Points')
plt.colorbar()
```

Out[13]: <matplotlib.colorbar.Colorbar at 0x1ef78e80310>



In [14]:

```
# Visualization: Scatter plot of input points with predictions
plt.subplot(1, 2, 2)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, cmap='viridis', la
plt.scatter(input_scaled[:, 0], input_scaled[:, 1], c=predictions,
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Route Similarity Prediction')
plt.colorbar()
plt.legend()

plt.tight_layout()
plt.show()

print("Predictions:", predictions)

```

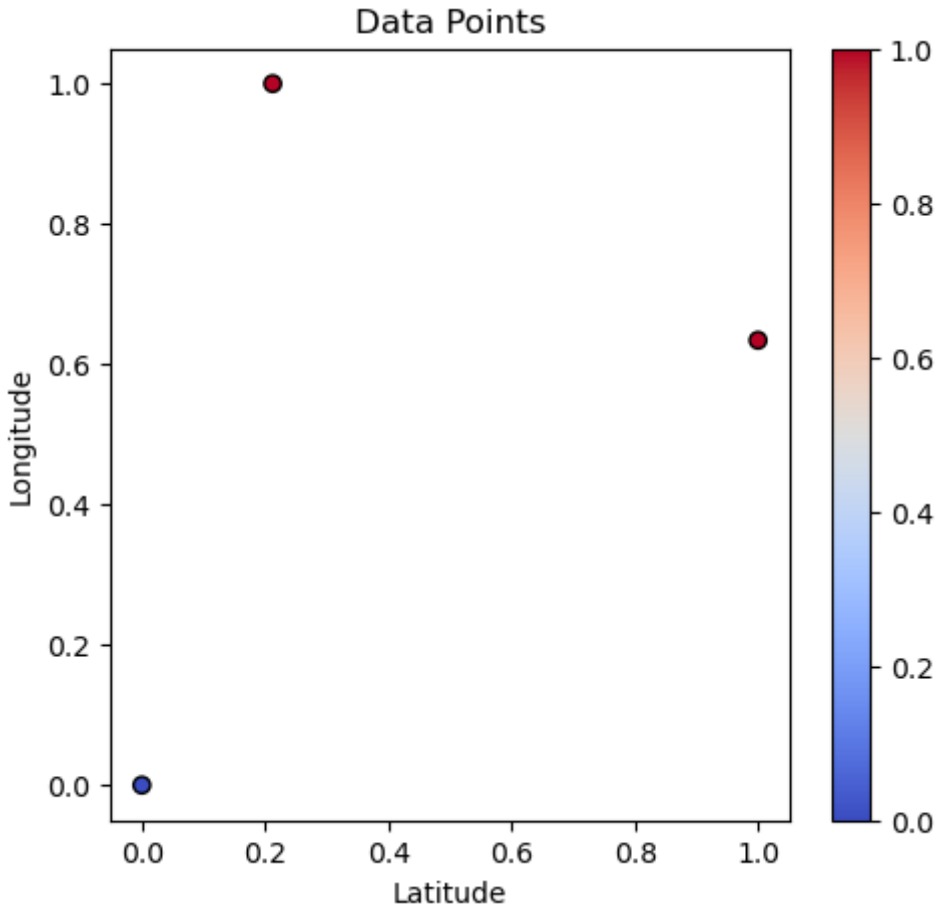



```
Predictions: [1. 1. 1.]
```

In [15]:

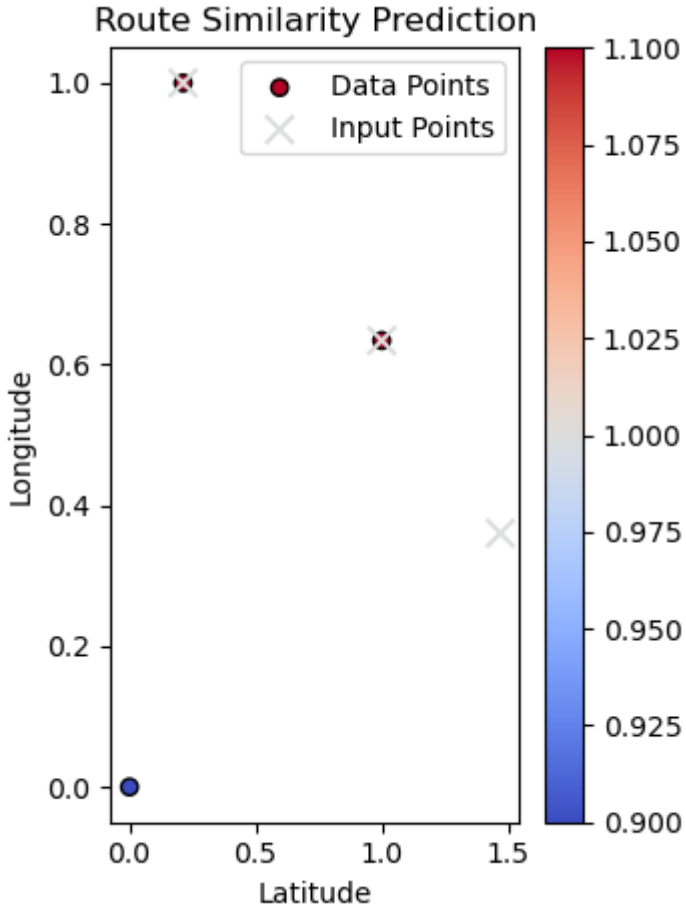
```
# Visualization: Scatter plot of data points
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, cmap='coolwarm', e
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Data Points')
plt.colorbar()
```

Out[15]: <matplotlib.colorbar.Colorbar at 0x1ef78fff0d0>



In [16]:

```
# Visualization: Scatter plot of input points with predictions
plt.subplot(1, 2, 2)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, cmap='coolwarm', e
plt.scatter(input_scaled[:, 0], input_scaled[:, 1], c=predictions,
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Route Similarity Prediction')
plt.colorbar()
plt.legend()

plt.tight_layout()
plt.show()

print("Predictions:", predictions)

```




```
Predictions: [1. 1. 1.]
```

In [17]:

```
'''the point are taken from the API we have created by connecting t
```

Out[17]:

```
'the point are taken from the API we have created by connecting to LOC
AL '
```

In [18]:

```python
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from geopy.distance import geodesic

# Example data: coordinates of bus stops and travel times (in minut
bus_stops = np.array([
    [17.2404, 78.4293],  # Shamshabad, Hyderabad
    [17.5453, 78.3408],  # Narkuda, Hyderabad
    [17.1582, 78.1874],  # Shabad, Hyderabad
    [17.7275, 78.2746],  # Rajiv Gandhi Airport
    [17.4265, 78.4585],  # Gandipet, Hyderabad
    [17.4646, 78.4859],  # Ocean Park, Hyderabad
    [17.4125, 78.4388],  # Mehdipatnam, Hyderabad
    [17.3986, 78.4703],  # Golconda Fort, Hyderabad
    [17.3822, 78.4867],  # Qutub Shahi Tombs, Hyderabad
    [17.3850, 78.4804],  # Toli Chowki, Hyderabad
    [17.4090, 78.4896],  # Banjara Hills, Hyderabad
    [17.4349, 78.4484],  # Film Nagar, Hyderabad
    [17.4396, 78.4170],  # HITEC City, Hyderabad
    [17.4461, 78.3490],  # JNTU, Hyderabad
    [17.4485, 78.3719],  # Kukatpally, Hyderabad
    [17.4531, 78.3636],  # Moosapet, Hyderabad
    [17.4711, 78.3656],  # Bharatnagar, Hyderabad
    [17.4861, 78.3850],  # Ameerpet, Hyderabad
    [17.5099, 78.3765],  # Punjagutta, Hyderabad
    [17.5184, 78.3893],  # Nagarjuna Circle, Hyderabad
    [17.4811, 78.4291],  # Masab Tank, Hyderabad
])

# Calculate distances between bus stops using geopy's geodesic
distances = [geodesic(bus_stops[i], bus_stops[i+1]).kilometers for

# Reshape distances for KNN input
X_knn = np.array(distances).reshape(-1, 1)
y_knn = bus_stops[1:]

# Standardize features for KNN
scaler_X_knn = StandardScaler()
X_scaled_knn = scaler_X_knn.fit_transform(X_knn)

# Train KNN model
knn_model = KNeighborsRegressor(n_neighbors=3)
knn_model.fit(X_scaled_knn, y_knn)

# Reshape distances for SVR input
X_svr = np.array(distances).reshape(-1, 1)
y_svr = travel_times[1:]

# Standardize features for SVR
scaler_X_svr = StandardScaler()
scaler_y_svr = StandardScaler()
X_scaled_svr = scaler_X_svr.fit_transform(X_svr)
y_scaled_svr = scaler_y_svr.fit_transform(y_svr.reshape(-1, 1))

# Train SVR model
svr_model = SVR(kernel='rbf')
svr_model.fit(X_scaled_svr, y_scaled_svr.ravel())

# Custom input distance
```

```
input_distance = geodesic(bus_stops[0], bus_stops[1]).kilometers

# Predict intermediate points using KNN
predicted_point = knn_model.predict([[input_distance]])
predicted_distance = np.array([geodesic(bus_stops[0], predicted_poi

# Scale predicted distance for SVR
predicted_distance_scaled = scaler_X_svr.transform(predicted_distan

# Predict travel time using SVR
predicted_time_scaled = svr_model.predict(predicted_distance_scaled
predicted_time = scaler_y_svr.inverse_transform(predicted_time_scal

# Print the predicted travel time
print(f"Predicted travel time to next bus stop: {predicted_time[0][

```

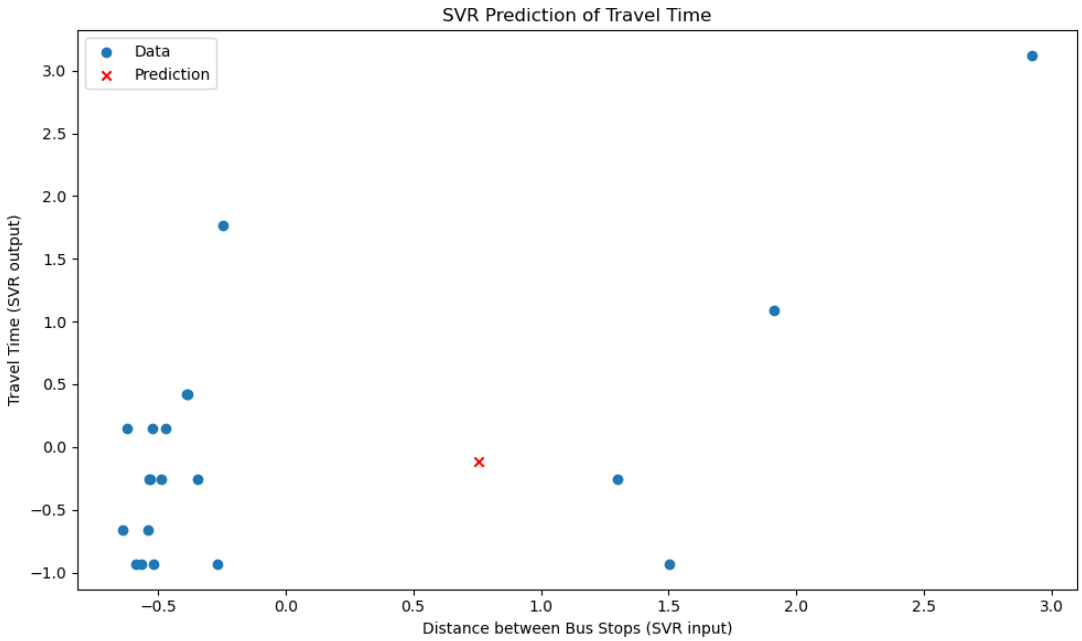
```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
~\AppData\Local\Temp\ipykernel_29236\2455553422.py in <module>
     47 # Reshape distances for SVR input
     48 X_svr = np.array(distances).reshape(-1, 1)
---> 49 y_svr = travel_times[1:]
     50
     51 # Standardize features for SVR

NameError: name 'travel_times' is not defined
```

In [19]:

```python
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from geopy.distance import geodesic

# Example data: coordinates of bus stops and travel times (in minut
bus_stops = np.array([
    [17.2404, 78.4293],  # Shamshabad, Hyderabad
    [17.5453, 78.3408],  # Narkuda, Hyderabad
    [17.1582, 78.1874],  # Shabad, Hyderabad
    [17.7275, 78.2746],  # Rajiv Gandhi Airport
    [17.4265, 78.4585],  # Gandipet, Hyderabad
    [17.4646, 78.4859],  # Ocean Park, Hyderabad
    [17.4125, 78.4388],  # Mehdipatnam, Hyderabad
    [17.3986, 78.4703],  # Golconda Fort, Hyderabad
    [17.3822, 78.4867],  # Qutub Shahi Tombs, Hyderabad
    [17.3850, 78.4804],  # Toli Chowki, Hyderabad
    [17.4090, 78.4896],  # Banjara Hills, Hyderabad
    [17.4349, 78.4484],  # Film Nagar, Hyderabad
    [17.4396, 78.4170],  # HITEC City, Hyderabad
    [17.4461, 78.3490],  # JNTU, Hyderabad
    [17.4485, 78.3719],  # Kukatpally, Hyderabad
    [17.4531, 78.3636],  # Moosapet, Hyderabad
    [17.4711, 78.3656],  # Bharatnagar, Hyderabad
    [17.4861, 78.3850],  # Ameerpet, Hyderabad
    [17.5099, 78.3765],  # Punjagutta, Hyderabad
    [17.5184, 78.3893],  # Nagarjuna Circle, Hyderabad
    [17.4811, 78.4291],  # Masab Tank, Hyderabad
])

bus_stop_names = [
    "Shamshabad", "Narkuda", "Shabad", "Rajiv Gandhi Airport", "Gar
    "Ocean Park", "Mehdipatnam", "Golconda Fort", "Qutub Shahi Tomb
    "Banjara Hills", "Film Nagar", "HITEC City", "JNTU", "Kukatpall
    "Moosapet", "Bharatnagar", "Ameerpet", "Punjagutta", "Nagarjuna
]

travel_times = np.array([0, 15, 25, 40, 10, 20, 30, 18, 15, 12, 10,

# Calculate distances between bus stops using geopy's geodesic
distances = [geodesic(bus_stops[i], bus_stops[i+1]).kilometers for

# Reshape distances for KNN input
X_knn = np.array(distances).reshape(-1, 1)
y_knn = bus_stops[1:]

# Standardize features for KNN
scaler_X_knn = StandardScaler()
X_scaled_knn = scaler_X_knn.fit_transform(X_knn)

# Train KNN model
knn_model = KNeighborsRegressor(n_neighbors=3)
knn_model.fit(X_scaled_knn, y_knn)

# Reshape distances for SVR input
X_svr = np.array(distances).reshape(-1, 1)
y_svr = travel_times[1:]

# Standardize features for SVR
```

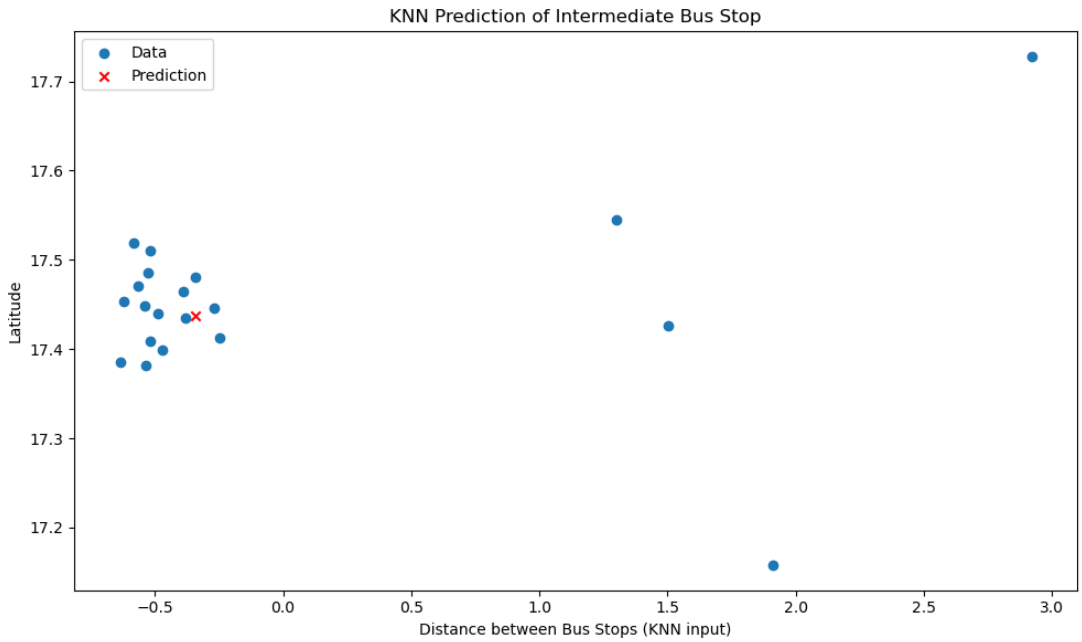
```
scaler_X_svr = StandardScaler()
scaler_y_svr = StandardScaler()
X_scaled_svr = scaler_X_svr.fit_transform(X_svr)
y_scaled_svr = scaler_y_svr.fit_transform(y_svr.reshape(-1, 1))

# Train SVR model
svr_model = SVR(kernel='rbf')
svr_model.fit(X_scaled_svr, y_scaled_svr.ravel())

# Custom input distance
source_stop_index = 0
destination_stop_index = 5
input_distance = geodesic(bus_stops[source_stop_index], bus_stops[d

# Predict intermediate point using KNN
predicted_point = knn_model.predict([[input_distance]])
predicted_distance = np.array([geodesic(bus_stops[source_stop_index

# Scale predicted distance for SVR
predicted_distance_scaled = scaler_X_svr.transform(predicted_distan

# Predict travel time using SVR
predicted_time_scaled = svr_model.predict(predicted_distance_scaled
predicted_time = scaler_y_svr.inverse_transform(predicted_time_scal

# Print the predicted travel time
print(f"Predicted travel time from {bus_stop_names[source_stop_inde


# Visualization: SVR prediction of travel time
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_svr, y_scaled_svr, label='Data')
plt.scatter(predicted_distance_scaled, predicted_time_scaled, color
plt.xlabel('Distance between Bus Stops (SVR input)')
plt.ylabel('Travel Time (SVR output)')
plt.title('SVR Prediction of Travel Time')
plt.legend()

plt.tight_layout()
plt.show()

```

```
Predicted travel time from Shamshabad to Ocean Park: 16.03 minutes
```



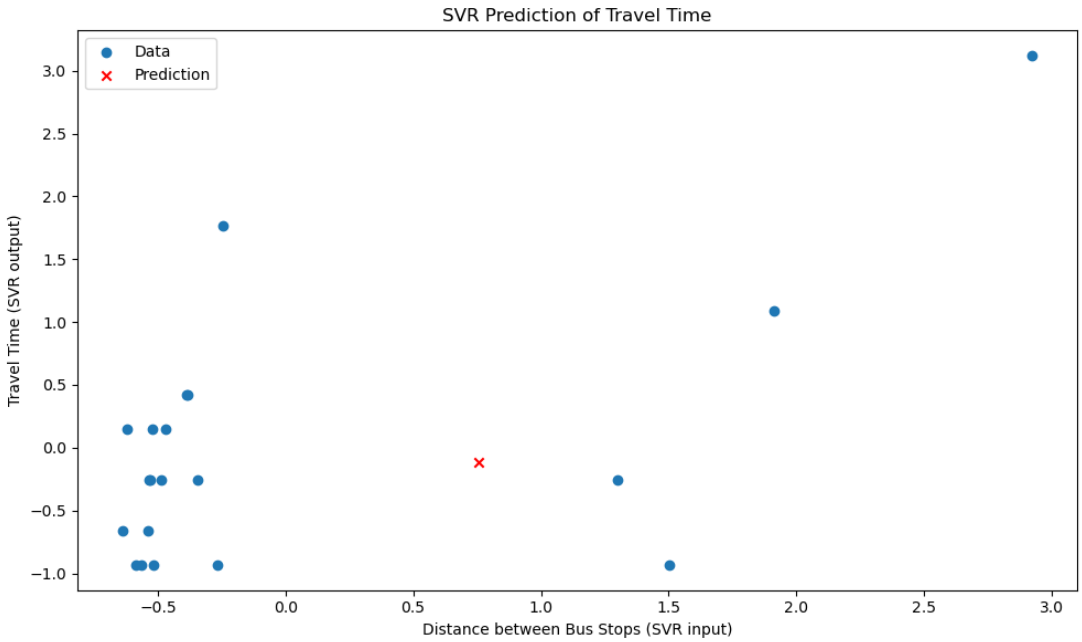
In [20]:

```
# Visualization: KNN prediction of intermediate point
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_knn, y_knn[:, 0], label='Data')
predicted_lat = knn_model.predict([[X_knn[-1][0]]])[0][0]
plt.scatter(X_scaled_knn[-1], predicted_lat, color='red', marker='x
plt.xlabel('Distance between Bus Stops (KNN input)')
plt.ylabel('Latitude')
plt.title('KNN Prediction of Intermediate Bus Stop')
plt.legend()

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



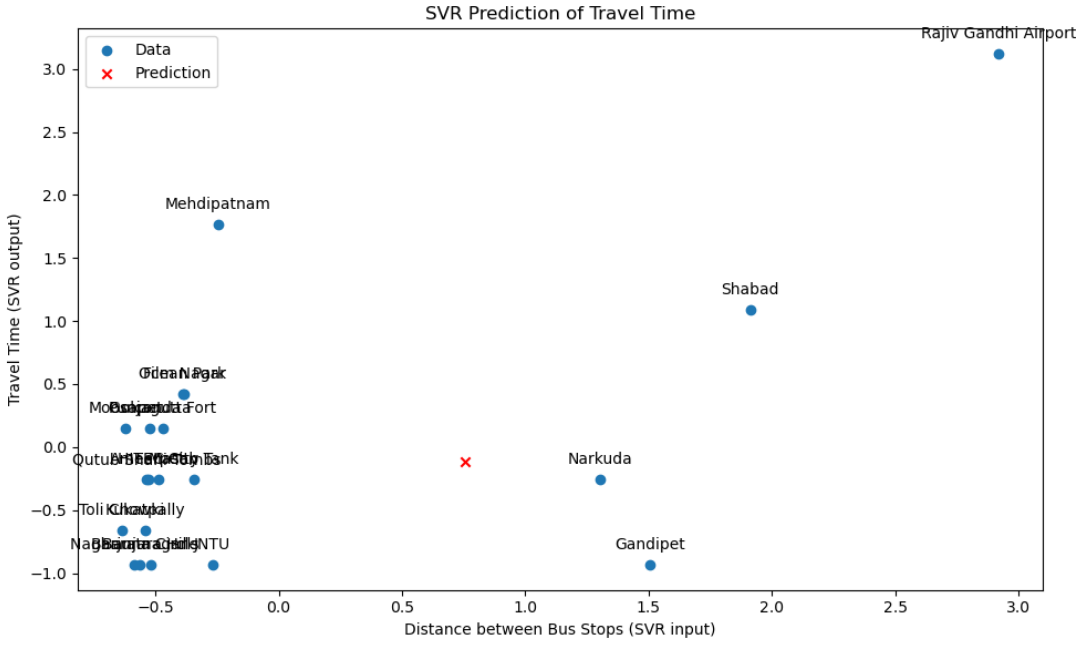
In [21]:

```
# Visualization: SVR prediction of travel time
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_svr, y_scaled_svr, label='Data')
plt.scatter(predicted_distance_scaled, predicted_time_scaled, color
plt.xlabel('Distance between Bus Stops (SVR input)')
plt.ylabel('Travel Time (SVR output)')
plt.title('SVR Prediction of Travel Time')
plt.legend()

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



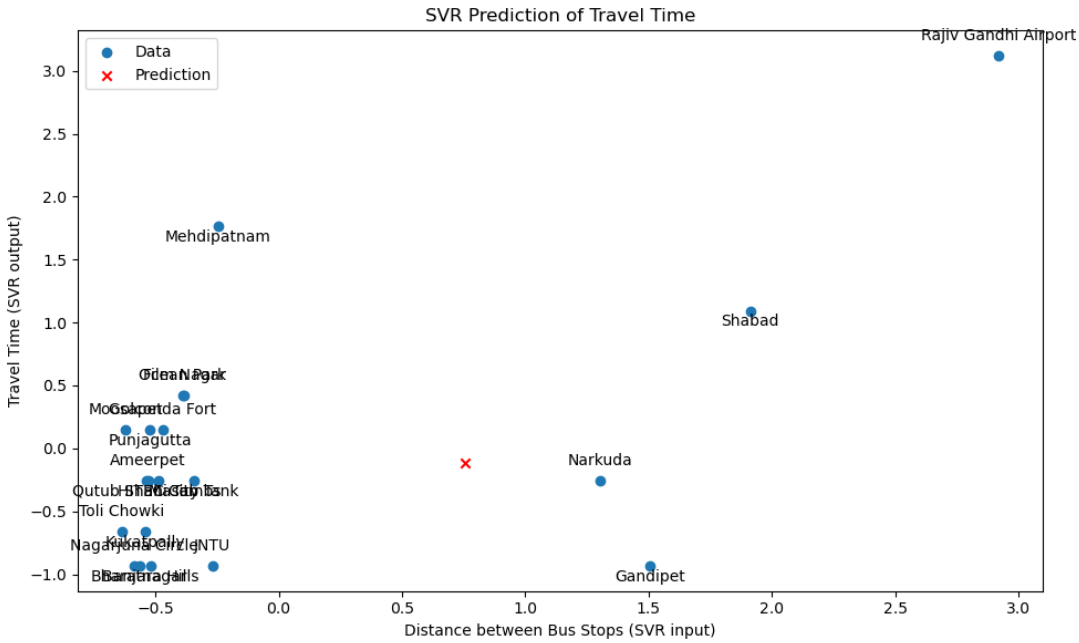
In [22]:

```
# Visualization: SVR prediction of travel time
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_svr, y_scaled_svr, label='Data')

# Add data labels for each point
for i, txt in enumerate(bus_stop_names[1:]):
    plt.annotate(txt, (X_scaled_svr[i], y_scaled_svr[i]), textcoord

# Add a red 'x' marker for the predicted point
plt.scatter(predicted_distance_scaled, predicted_time_scaled, color

plt.xlabel('Distance between Bus Stops (SVR input)')
plt.ylabel('Travel Time (SVR output)')
plt.title('SVR Prediction of Travel Time')
plt.legend()

plt.tight_layout()
plt.show()

```



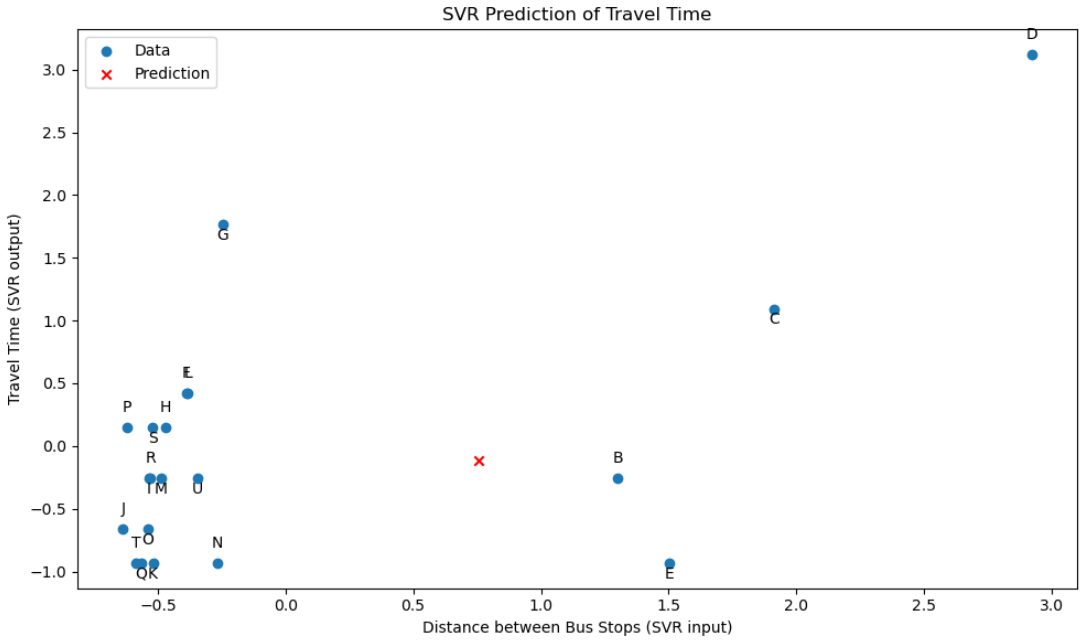
In [23]:

```
# Visualization: SVR prediction of travel time
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_svr, y_scaled_svr, label='Data')

# Add data labels for each point with adjusted positioning
for i, txt in enumerate(bus_stop_names[1:]):
    if i % 2 == 0:
        plt.annotate(txt, (X_scaled_svr[i], y_scaled_svr[i]), textc
    else:
        plt.annotate(txt, (X_scaled_svr[i], y_scaled_svr[i]), textc

# Add a red 'x' marker for the predicted point
plt.scatter(predicted_distance_scaled, predicted_time_scaled, color

plt.xlabel('Distance between Bus Stops (SVR input)')
plt.ylabel('Travel Time (SVR output)')
plt.title('SVR Prediction of Travel Time')
plt.legend()

plt.tight_layout()
plt.show()

```



In [24]:

```
# Short names for bus stops
bus_stop_short_names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'

# Visualization: SVR prediction of travel time
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_svr, y_scaled_svr, label='Data')

# Add data labels for each point with short names
for i, txt in enumerate(bus_stop_short_names[1:]):
    if i % 2 == 0:
        plt.annotate(txt, (X_scaled_svr[i], y_scaled_svr[i]), textc
    else:
        plt.annotate(txt, (X_scaled_svr[i], y_scaled_svr[i]), textc

# Add a red 'x' marker for the predicted point
plt.scatter(predicted_distance_scaled, predicted_time_scaled, color

plt.xlabel('Distance between Bus Stops (SVR input)')
plt.ylabel('Travel Time (SVR output)')
plt.title('SVR Prediction of Travel Time')
plt.legend()

plt.tight_layout()
plt.show()

```



In [1]:

```python
import pygame

# Initialize pygame
pygame.init()

# Load a sound file
sound_file = "mixkit-game-show-suspense-waiting-667.wav"
pygame.mixer.music.load(sound_file)

# Play the sound
pygame.mixer.music.play()

# Wait for the sound to finish playing
while pygame.mixer.music.get_busy():
    pygame.time.Clock().tick(10)

# Quit pygame
pygame.quit()

```

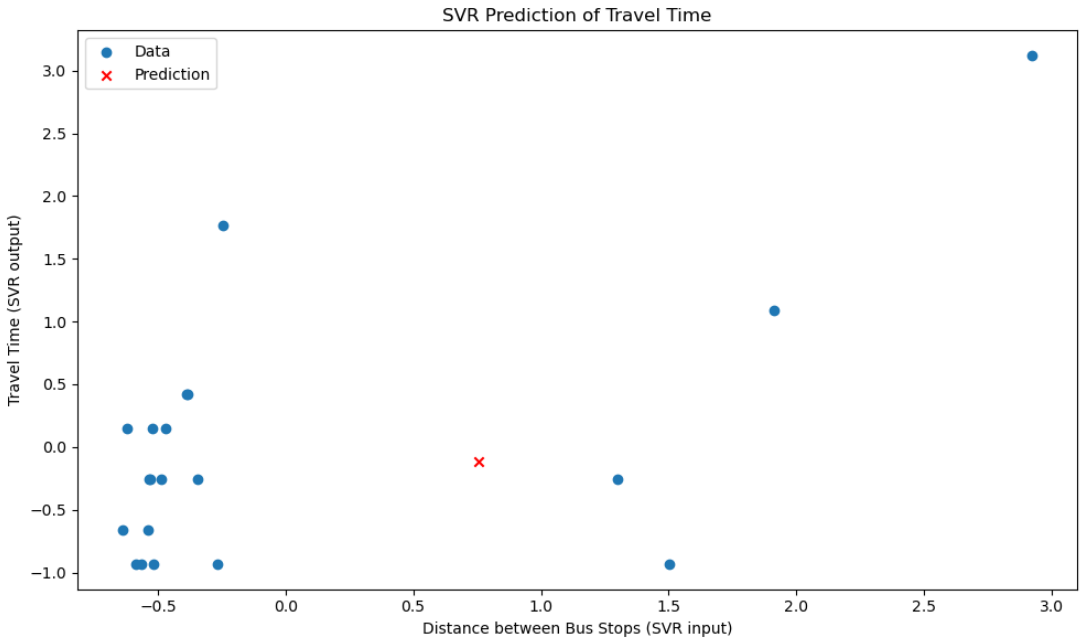
```
pygame 2.5.1 (SDL 2.28.2, Python 3.9.13)
Hello from the pygame community. https://www.pygame.org/contribute.html (https://www.pygame.org/contribute.html)
```

In [27]:

```python
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from geopy.distance import geodesic

# Example data: coordinates of bus stops and travel times (in minut
bus_stops = np.array([
    [17.2404, 78.4293],  # Shamshabad, Hyderabad
    [17.5453, 78.3408],  # Narkuda, Hyderabad
    [17.1582, 78.1874],  # Shabad, Hyderabad
    [17.7275, 78.2746],  # Rajiv Gandhi Airport
    [17.4265, 78.4585],  # Gandipet, Hyderabad
    [17.4646, 78.4859],  # Ocean Park, Hyderabad
    [17.4125, 78.4388],  # Mehdipatnam, Hyderabad
    [17.3986, 78.4703],  # Golconda Fort, Hyderabad
    [17.3822, 78.4867],  # Qutub Shahi Tombs, Hyderabad
    [17.3850, 78.4804],  # Toli Chowki, Hyderabad
    [17.4090, 78.4896],  # Banjara Hills, Hyderabad
    [17.4349, 78.4484],  # Film Nagar, Hyderabad
    [17.4396, 78.4170],  # HITEC City, Hyderabad
    [17.4461, 78.3490],  # JNTU, Hyderabad
    [17.4485, 78.3719],  # Kukatpally, Hyderabad
    [17.4531, 78.3636],  # Moosapet, Hyderabad
    [17.4711, 78.3656],  # Bharatnagar, Hyderabad
    [17.4861, 78.3850],  # Ameerpet, Hyderabad
    [17.5099, 78.3765],  # Punjagutta, Hyderabad
    [17.5184, 78.3893],  # Nagarjuna Circle, Hyderabad
    [17.4811, 78.4291],  # Masab Tank, Hyderabad
])

bus_stop_names = [
    "Shamshabad", "Narkuda", "Shabad", "Rajiv Gandhi Airport", "Gar
    "Ocean Park", "Mehdipatnam", "Golconda Fort", "Qutub Shahi Tomb
    "Banjara Hills", "Film Nagar", "HITEC City", "JNTU", "Kukatpall
    "Moosapet", "Bharatnagar", "Ameerpet", "Punjagutta", "Nagarjuna
]

travel_times = np.array([0, 15, 25, 40, 10, 20, 30, 18, 15, 12, 10,

# Calculate distances between bus stops using geopy's geodesic
distances = [geodesic(bus_stops[i], bus_stops[i+1]).kilometers for

# Reshape distances for KNN input
X_knn = np.array(distances).reshape(-1, 1)
y_knn = bus_stops[1:]

# Standardize features for KNN
scaler_X_knn = StandardScaler()
X_scaled_knn = scaler_X_knn.fit_transform(X_knn)

# Train KNN model
knn_model = KNeighborsRegressor(n_neighbors=3)
knn_model.fit(X_scaled_knn, y_knn)

# Reshape distances for SVR input
X_svr = np.array(distances).reshape(-1, 1)
y_svr = travel_times[1:]

# Standardize features for SVR
```

```
scaler_X_svr = StandardScaler()
scaler_y_svr = StandardScaler()
X_scaled_svr = scaler_X_svr.fit_transform(X_svr)
y_scaled_svr = scaler_y_svr.fit_transform(y_svr.reshape(-1, 1))

# Train SVR model
svr_model = SVR(kernel='rbf')
svr_model.fit(X_scaled_svr, y_scaled_svr.ravel())

# Custom input distance
source_stop_index = 0
destination_stop_index = 5
input_distance = geodesic(bus_stops[source_stop_index], bus_stops[d

# Predict intermediate point using KNN
predicted_point = knn_model.predict([[input_distance]])
predicted_distance = np.array([geodesic(bus_stops[source_stop_index

# Scale predicted distance for SVR
predicted_distance_scaled = scaler_X_svr.transform(predicted_distan

# Predict travel time using SVR
predicted_time_scaled = svr_model.predict(predicted_distance_scaled
predicted_time = scaler_y_svr.inverse_transform(predicted_time_scal

# Print the predicted travel time
print(f"Predicted travel time from {bus_stop_names[source_stop_inde


# Visualization: SVR prediction of travel time
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_svr, y_scaled_svr, label='Data')
plt.scatter(predicted_distance_scaled, predicted_time_scaled, color
plt.xlabel('Distance between Bus Stops (SVR input)')
plt.ylabel('Travel Time (SVR output)')
plt.title('SVR Prediction of Travel Time')
plt.legend()

plt.tight_layout()
plt.show()

```

```
Predicted travel time from Shamshabad to Ocean Park: 16.03 minutes
```


SVR Prediction of Travel Time
Data
Prediction
Travel Time (SVR output)
Distance between Bus Stops (SVR input)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

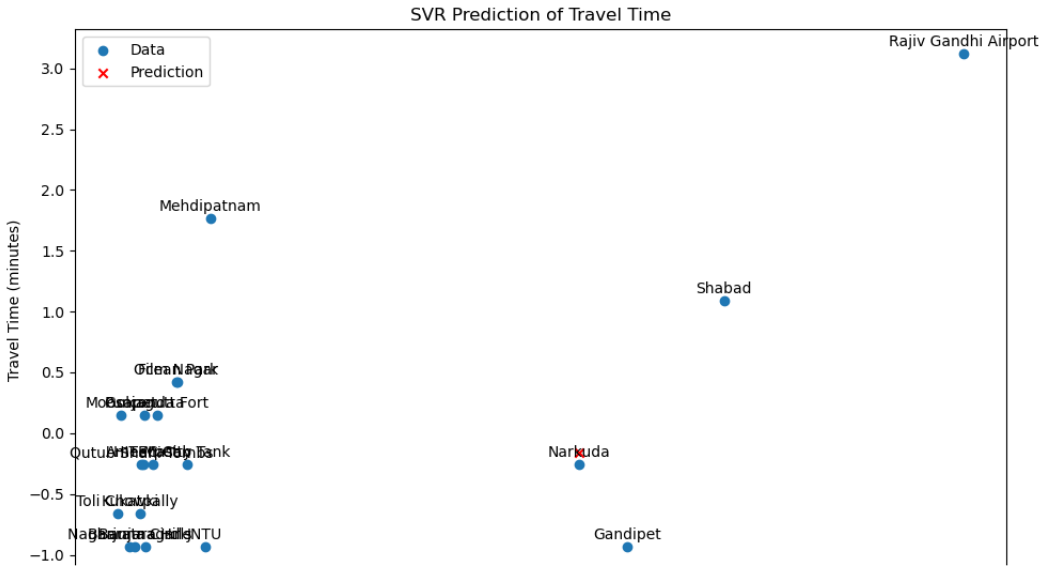
In [29]:

```python
import numpy as np
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from geopy.distance import geodesic
import time
import pygame

# Example data: coordinates of bus stops and travel times (in minut
bus_stops = np.array([
    [17.2404, 78.4293],  # Shamshabad, Hyderabad
    [17.5453, 78.3408],  # Narkuda, Hyderabad
    [17.1582, 78.1874],  # Shabad, Hyderabad
    [17.7275, 78.2746],  # Rajiv Gandhi International Airport (not
    [17.4265, 78.4585],  # Gandipet, Hyderabad
    [17.4646, 78.4859],  # Ocean Park, Hyderabad
    [17.4125, 78.4388],  # Mehdipatnam, Hyderabad
    [17.3986, 78.4703],  # Golconda Fort, Hyderabad
    [17.3822, 78.4867],  # Qutub Shahi Tombs, Hyderabad
    [17.3850, 78.4804],  # Toli Chowki, Hyderabad
    [17.4090, 78.4896],  # Banjara Hills, Hyderabad
    [17.4349, 78.4484],  # Film Nagar, Hyderabad
    [17.4396, 78.4170],  # HITEC City, Hyderabad
    [17.4461, 78.3490],  # JNTU, Hyderabad
    [17.4485, 78.3719],  # Kukatpally, Hyderabad
    [17.4531, 78.3636],  # Moosapet, Hyderabad
    [17.4711, 78.3656],  # Bharatnagar, Hyderabad
    [17.4861, 78.3850],  # Ameerpet, Hyderabad
    [17.5099, 78.3765],  # Punjagutta, Hyderabad
    [17.5184, 78.3893],  # Nagarjuna Circle, Hyderabad
    [17.4811, 78.4291],  # Masab Tank, Hyderabad
])

bus_stop_names = [
    "Shamshabad", "Narkuda", "Shabad", "Rajiv Gandhi Airport", "Gar
    "Ocean Park", "Mehdipatnam", "Golconda Fort", "Qutub Shahi Tomb
    "Banjara Hills", "Film Nagar", "HITEC City", "JNTU", "Kukatpall
    "Moosapet", "Bharatnagar", "Ameerpet", "Punjagutta", "Nagarjuna
]

travel_times = np.array([0, 15, 25, 40, 10, 20, 30, 18, 15, 12, 10,

# Calculate distances between bus stops using geopy's geodesic
distances = [geodesic(bus_stops[i], bus_stops[i+1]).kilometers for

# Reshape distances and travel times for SVR input
X = np.array(distances).reshape(-1, 1)
y = travel_times[1:]

# Standardize features
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.reshape(-1, 1))

# Train SVR model
svr_model = SVR(kernel='rbf')
svr_model.fit(X_scaled, y_scaled.ravel())  # Ravel y_scaled to make

# Custom input distance
input_distance = geodesic(bus_stops[0], bus_stops[1]).kilometers
```

```
62
63  # Scale input distance
64  input_distance_scaled = scaler_X.transform(np.array([[input_distanc
65
66  # Predict travel time using SVR
67  predicted_time_scaled = svr_model.predict(input_distance_scaled)
68  predicted_time = scaler_y.inverse_transform(predicted_time_scaled.r
69
70  # Print the predicted travel time
71  print(f"Predicted travel time to next bus stop: {predicted_time[0][
72
73  # Print the predicted travel time
74  source_stop_name = bus_stop_names[source_stop_index]
75  destination_stop_name = bus_stop_names[destination_stop_index]
76  print(f"Predicted travel time from {source_stop_name} to {destinati
77
78  # Play an alarm sound 2 minutes before reaching the predicted trave
79  alarm_time = predicted_time[0][0] - 2  # 2 minutes before predicted
80  current_time = time.time()
81  time_to_wait = alarm_time * 60 - current_time  # Convert to seconds
82
83  # Play the alarm sound 2 minutes before reaching the destination
84  if time_to_wait > 0:
85      time.sleep(time_to_wait)
86
87      # Initialize Pygame mixer
88      pygame.mixer.init()
89
90      # Load and play the alarm sound
91      pygame.mixer.music.load("mixkit-game-show-suspense-waiting-667.
92      pygame.mixer.music.play()
93
94      # Wait for the sound to finish playing
95      while pygame.mixer.music.get_busy():
96          pygame.time.Clock().tick(10)
97
98  # Visualization: SVR prediction
99  plt.figure(figsize=(10, 6))
100 plt.scatter(X_scaled, y_scaled, label='Data')
101 plt.scatter(input_distance_scaled, predicted_time_scaled, color='re
102 plt.xlabel('Distance between Bus Stops')
103 plt.ylabel('Travel Time (minutes)')
104 plt.title('SVR Prediction of Travel Time')
105 plt.legend()
106
107 # Annotate bus stop names on the graph
108 for i, name in enumerate(bus_stop_names[1:]):
109     plt.annotate(name, (X_scaled[i][0], y_scaled[i][0]), textcoords
110
111 plt.tight_layout()
112 plt.show()
113
114 # Visualization: Original data points
115 plt.figure(figsize=(10, 6))
116 plt.scatter(X, y)
117 plt.xlabel('Distance between Bus Stops')
118 plt.ylabel('Travel Time (minutes)')
119 plt.title('Original Data Points')
120 plt.show()
121
122 # Visualization: SVR prediction with bus stop names
```

```
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled, y_scaled, label='Data')
plt.scatter(input_distance_scaled, predicted_time_scaled, color='re
plt.xlabel('Distance between Bus Stops')
plt.ylabel('Travel Time (minutes)')
plt.title('SVR Prediction of Travel Time')
plt.legend()

# Annotate bus stop names on the graph
for i, name in enumerate(bus_stop_names[1:]):
    plt.annotate(name, (X_scaled[i][0], y_scaled[i][0]), textcoords

plt.tight_layout()
plt.show()

# Visualization: Predicted travel time vs. Actual travel time
plt.figure(figsize=(10, 6))
plt.plot(y_scaled, label='Actual Travel Time')
plt.plot(predicted_time_scaled, label='Predicted Travel Time')
plt.xlabel('Bus Stop')
plt.ylabel('Normalized Travel Time')
plt.title('Predicted Travel Time vs. Actual Travel Time')
plt.legend()
plt.xticks(range(len(bus_stop_names[1:])), bus_stop_names[1:], rota
plt.tight_layout()
plt.show()
```

```
Predicted travel time to next bus stop: 15.74 minutes
Predicted travel time from Shamshabad to Ocean Park: 15.74 minutes
```



In [ ]:
```
'''Above one is complete code'''
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```