In [1]:
```python
#importing all required python libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt #use to visualize dataset values
import seaborn as sns
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score
from sklearn import svm
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
import pandas as pd
import os
import pickle
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.cluster import DBSCAN#Loading DBSCAN clustering
from sklearn.neighbors import KernelDensity #Loading kernel density algorithms
from sklearn.decomposition import PCA #pca for dimension reduction
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
```

In [2]:
```python
#class to normalize dataset values
scaler = MinMaxScaler(feature_range = (0, 1))
scaler1 = MinMaxScaler(feature_range = (0, 1))
```

In [3]:
```python
#loading and displaying cellular lte dataset
dataset = pd.read_csv("Dataset/SRFG-v1.csv", nrows=10000)
dataset.head()
```

Out[3]:

	time	lat	long	ele	newpos	rsrq	cell_id	sinr	signal	pci	...	technology	rsrp	datarate	file	line	trip	predecessor	dlong	dlat	predecessor2
0	2018-01-15 16:43:52	47.847847	13.080113	505.1	True	-6.0	798470	28.0	5.0	468.0	...	LTE	-65.0	5554624.0	2018-01-15-1736	417	2018-01-15-1736	NaN	NaN	NaN	NaN
1	2018-01-15 16:43:53	47.848070	13.080318	505.2	True	-7.0	798470	24.0	5.0	468.0	...	LTE	-68.0	4814816.0	2018-01-15-1736	418	2018-01-15-1736	1.0	0.000205	0.000221	NaN
2	2018-01-15 16:43:54	47.848280	13.080542	505.5	True	-7.0	798470	22.0	5.0	468.0	...	LTE	-67.0	5081632.0	2018-01-15-1736	419	2018-01-15-1736	1.0	0.000223	0.000214	NaN
3	2018-01-15 16:43:55	47.848484	13.080778	505.7	True	-6.0	798470	22.0	5.0	468.0	...	LTE	-70.0	5506112.0	2018-01-15-1736	420	2018-01-15-1736	1.0	0.000237	0.000202	NaN
4	2018-01-15 16:43:56	47.848680	13.081034	505.8	True	-7.0	798470	17.0	5.0	468.0	...	LTE	-77.0	5518240.0	2018-01-15-1736	421	2018-01-15-1736	1.0	0.000256	0.000195	NaN

5 rows × 22 columns

In [4]:
```python
#dataset peprocessing converting datetime to numeric values and non-numeric values to numeric values
dataset['time'] = pd.to_datetime(dataset['time'])
dataset['year'] = dataset['time'].dt.year
dataset['month'] = dataset['time'].dt.month
dataset['day'] = dataset['time'].dt.day
dataset['hour'] = dataset['time'].dt.hour
dataset['minute'] = dataset['time'].dt.minute
dataset['second'] = dataset['time'].dt.second
label_encoder = []
columns = dataset.columns
types = dataset.dtypes.values
for i in range(len(types)):
    name = types[i]
    if name == 'object': #finding column with object type
        le = LabelEncoder()
        dataset[columns[i]] = pd.Series(le.fit_transform(dataset[columns[i]].astype(str)))#encode all str columns to numeric
        label_encoder.append([columns[i], le])
#handling and removing missing values
dataset.fillna(0, inplace = True)
print("Cleaned Dataset Values")
dataset
```

```
Cleaned Dataset Values
```

Out[4]:

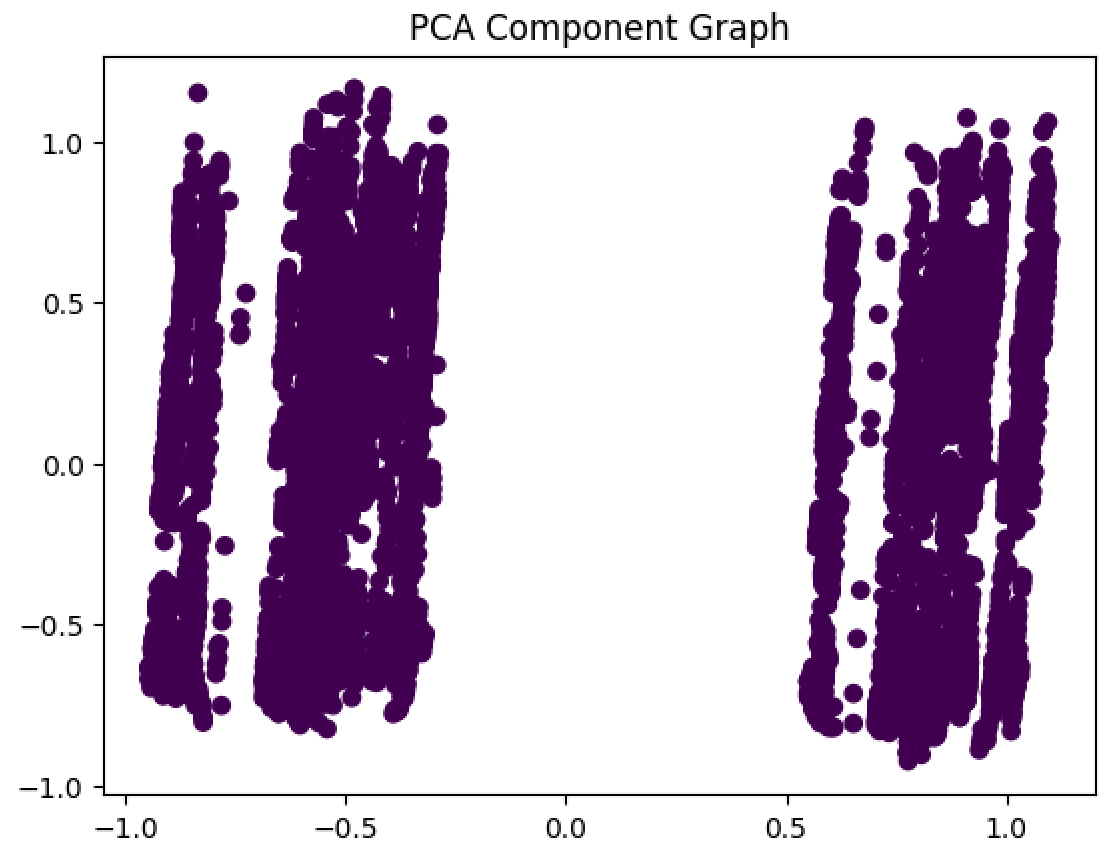
	time	lat	long	ele	newpos	rsrq	cell_id	sinr	signal	pci	...	predecessor	dlong	dlat	predecessor2	year	month	day	hour	minute	second
0	2018-01-15 16:43:52	47.847847	13.080113	505.1	True	-6.0	798470	28.0	5.0	468.0	...	0.0	0.000000	0.000000	0.0	2018	1	15	16	43	52
1	2018-01-15 16:43:53	47.848070	13.080318	505.2	True	-7.0	798470	24.0	5.0	468.0	...	1.0	0.000205	0.000221	0.0	2018	1	15	16	43	53
2	2018-01-15 16:43:54	47.848280	13.080542	505.5	True	-7.0	798470	22.0	5.0	468.0	...	1.0	0.000223	0.000214	0.0	2018	1	15	16	43	54
3	2018-01-15 16:43:55	47.848484	13.080778	505.7	True	-6.0	798470	22.0	5.0	468.0	...	1.0	0.000237	0.000202	0.0	2018	1	15	16	43	55
4	2018-01-15 16:43:56	47.848680	13.081034	505.8	True	-7.0	798470	17.0	5.0	468.0	...	1.0	0.000256	0.000195	0.0	2018	1	15	16	43	56
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	2018-02-06 16:29:10	47.843945	13.295335	555.1	True	-16.0	421638	-3.0	1.0	198.0	...	1.0	0.000424	0.000019	0.0	2018	2	6	16	29	10
9996	2018-02-06 16:29:11	47.843964	13.295760	555.2	True	-16.0	421638	-3.0	1.0	198.0	...	1.0	0.000425	0.000019	0.0	2018	2	6	16	29	11
9997	2018-02-06 16:29:12	47.843983	13.296185	555.4	True	-14.0	421638	-2.0	1.0	198.0	...	1.0	0.000424	0.000019	0.0	2018	2	6	16	29	12
9998	2018-02-06 16:29:13	47.844000	13.296608	555.6	True	-16.0	421638	-4.0	1.0	198.0	...	1.0	0.000423	0.000019	0.0	2018	2	6	16	29	13
9999	2018-02-06 16:29:14	47.844020	13.297033	555.7	True	-18.0	421638	-8.0	1.0	198.0	...	1.0	0.000425	0.000019	0.0	2018	2	6	16	29	14

10000 rows × 28 columns

In [5]:
```python
Y = dataset['netmode'].ravel()
Y1 = dataset['datarate'].ravel()
dataset.drop(['time','netmode', 'datarate'], axis = 1,inplace=True)#drop ir-relevant columns
X = dataset.values
X = scaler.fit_transform(X)#normalizing dataset values using minmax scaling
selector = SelectKBest(chi2, k=20)#selecting top 20 features using Select k BEST
X = selector.fit_transform(X, Y)
#applying PCA for dimension reduction
pca = PCA(n_components=15)
X = pca.fit_transform(X)
print("PCA Selected features = "+str(X))
```

```
PCA Selected features = [[-8.40036376e-01 -7.50821105e-01 -1.09147217e-01 ...  5.44867410e-02
   1.50901200e-01 -5.75822244e-02]
 [-8.84150193e-01 -7.29006560e-01  2.50242144e-02 ...  9.08868760e-03
  -3.61587519e-05 -5.27387833e-02]
 [-8.87518987e-01 -7.23276161e-01  3.75553518e-02 ... -1.33755190e-02
  -1.24752472e-02 -2.18834424e-02]
 ...
 [ 7.77155793e-01  5.26586108e-01  9.60360811e-01 ...  1.64084410e-01
   4.59808043e-02 -6.91956428e-02]
 [ 7.79330896e-01  5.67191597e-01  9.62805947e-01 ...  1.09302138e-01
   4.80241597e-02 -1.10009959e-01]
 [ 7.81378080e-01  6.23998914e-01  9.68590263e-01 ...  4.17893293e-02
   4.56368499e-02 -1.01851015e-01]]
```

In [6]:
```
plot = plt.scatter(X[:,0], X[:,1], c=Y)
plt.title("PCA Component Graph")
plt.show()
```



In [7]:
```
#applying DBSCAN based clustering with kernel density to select cluster with most similarity
dbscan = DBSCAN(eps=0.9, min_samples=8)#generating DBSCAN clustering
labels = dbscan.fit_predict(X)
# Kernel Density Estimation for each cluster
unique_labels = np.unique(labels)
choosen_cluster = []
choosen_labels = []
for label in unique_labels:
    if label == -1:  # Noise points
        continue
    cluster_data = X[labels == label]
    YY = Y1[labels == label]
    kde = KernelDensity(bandwidth=0.5).fit(cluster_data)#applying kernel density
    density = kde.score_samples(cluster_data)
    choosen_cluster.append(cluster_data)
    choosen_labels.append(YY)
similarity = 100000
selected = -1
for i in range(0, len(choosen_cluster)):
    cluster1 = choosen_cluster[i]
    for j in range(0, len(choosen_cluster)):
        if i != j:
            cluster2 = choosen_cluster[j]
            sim = cosine_similarity(cluster1, cluster2)#measuring similarity between clusters
            if np.mean(sim) < similarity:
                similarity = np.mean(sim)
                selected = i
X = choosen_cluster[selected]
Y = choosen_labels[selected]
print("Number of selected Clusters = "+str(len(choosen_cluster)))
print("Cluster values with most similarity = "+str(X))
```

```
Number of selected Clusters = 2
Cluster values with most similarity = [[-8.40036376e-01 -7.50821105e-01 -1.09147217e-01 ...  5.44867410e-02
   1.50901200e-01 -5.75822244e-02]
 [-8.84150193e-01 -7.29006560e-01  2.50242144e-02 ...  9.08868760e-03
  -3.61587519e-05 -5.27387833e-02]
 [-8.87518987e-01 -7.23276161e-01  3.75553518e-02 ... -1.33755190e-02
  -1.24752472e-02 -2.18834424e-02]
 ...
 [-5.57262529e-01 -6.08370273e-02  5.71791199e-01 ... -5.92058525e-02
   2.38358446e-01  2.50879267e-02]
 [-5.46656000e-01  6.56272073e-02  5.71974238e-01 ...  1.08118452e-01
   2.53522415e-01 -5.87609381e-02]
 [-5.45304410e-01  1.11615112e-01  5.79932244e-01 ... -1.32480869e-02
   2.46743849e-01  4.09950587e-02]]
```

In [8]:
```
scaler1 = StandardScaler()

# Reshape and scale Y
Y = Y.reshape(-1, 1)
Y = scaler1.fit_transform(Y)

# File paths
X_path = 'model/X.npy'
Y_path = 'model/Y.npy'

# Check if saved arrays exist
if os.path.exists(X_path) and os.path.exists(Y_path):
    # Load saved arrays
    X = np.load(X_path)
    Y = np.load(Y_path)
    print("Loaded pre-existing X and Y data.")
else:
    # Save arrays if not found
    np.save(X_path, X)
    np.save(Y_path, Y)
    print("Saved X and Y data.")

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Print information about the split data
print("Train & Test Dataset Split")
print("Total records found in selected cluster =", X.shape[0])
print("Total features found in selected cluster =", X.shape[1])
print("80% records used to train algorithms:", X_train.shape[0])
print("20% records used to test algorithms:", X_test.shape[0])
```

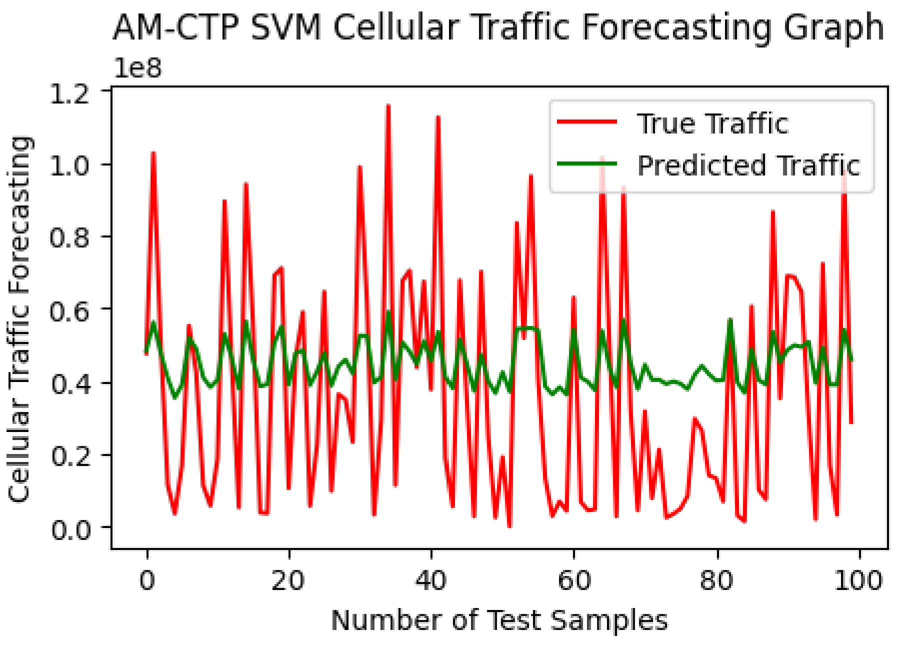
```
Loaded pre-existing X and Y data.
Train & Test Dataset Split
Total records found in selected cluster = 6051
Total features found in selected cluster = 15
80% records used to train algorithms: 4840
20% records used to test algorithms: 1211
```

In [9]:
```
#defining global variables to save algorithm performnace metrics
rsquare = []
mape = []
rmse = []
```

In [10]:
```
#function to calculate MAPE, RMSE and R2Square from predicted and true values
def calculateMetrics(algorithm, predict, test_labels):
    mape_error = mean_absolute_error(test_labels, predict)
    r2_scores = r2_score(np.asarray(test_labels), np.asarray(predict))
    rmse_error = sqrt(mean_squared_error(test_labels, predict))
    rsquare.append(r2_scores)
    mape.append(mape_error)
    rmse.append(rmse_error)
    predict = predict.reshape(-1, 1)
    predict = scaler1.inverse_transform(predict)
    test_label = scaler1.inverse_transform(test_labels)
    predict = predict.ravel()
    test_label = test_label.ravel()
    print()
    print(algorithm+" MAE : "+str(mape_error))
    print(algorithm+" RMSE : "+str(rmse_error))
    print(algorithm+" R2 : "+str(r2_scores))
    print()
    for i in range(0, 10):
        print("True Cellular Traffic : "+str(test_label[i])+" Predicted Cellular Traffic : "+str(predict[i]))
    plt.figure(figsize=(5,3))
    plt.plot(test_label[0:100], color = 'red', label = 'True Traffic')
    plt.plot(predict[0:100], color = 'green', label = 'Predicted Traffic')
    plt.title(algorithm+' Cellular Traffic Forecasting Graph')
    plt.xlabel('Number of Test Samples')
    plt.ylabel('Cellular Traffic Forecasting')
    plt.legend()
    plt.show()
```

In [11]:
```
#training propose AM-CTP SVM algorithm
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1)

# Path to save the model
model_path = 'model/svm_model.pkl'

# Check if the model file exists
if os.path.exists(model_path):
    # Load the pre-trained SVM model
    with open(model_path, 'rb') as f:
        svm_cls = pickle.load(f)
    print("Loaded pre-existing SVM model.")
else:
    # Initialize and train the SVM model
    svm_cls = svm.SVR()
    svm_cls.fit(X_train, y_train.ravel())

    # Save the trained model to file
    with open(model_path, 'wb') as f:
        pickle.dump(svm_cls, f)
    print("Trained and saved new SVM model.")

# Perform prediction on the test data
predict = svm_cls.predict(X_test)

# Call this function to calculate performance metrics
calculateMetrics("AM-CTP SVM", predict, y_test)
```

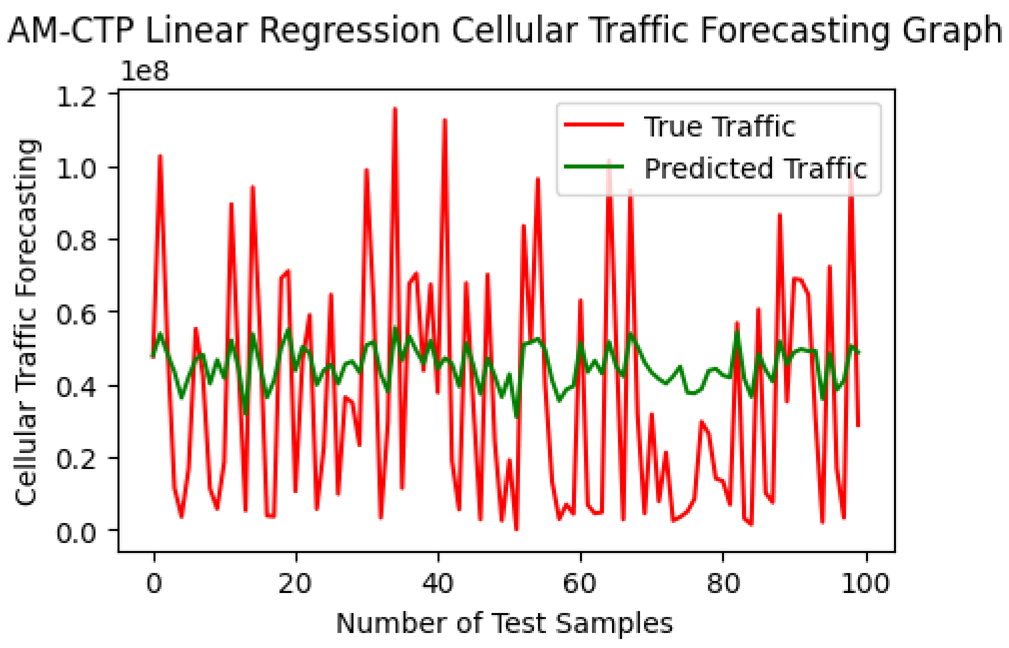
```
Loaded pre-existing SVM model.

AM-CTP SVM MAE : 0.7593321128912345
AM-CTP SVM RMSE : 0.8654772082973776
AM-CTP SVM R2 : 0.24792794003852636

True Cellular Traffic : 47711330.0 Predicted Cellular Traffic : 48454008.55340689
True Cellular Traffic : 102687780.0 Predicted Cellular Traffic : 56160849.07650396
True Cellular Traffic : 51632640.0 Predicted Cellular Traffic : 48003297.2196766
True Cellular Traffic : 11533728.0 Predicted Cellular Traffic : 41004110.861858524
True Cellular Traffic : 3565631.9999999963 Predicted Cellular Traffic : 35470751.75435179
True Cellular Traffic : 16833664.0 Predicted Cellular Traffic : 38996571.75417322
True Cellular Traffic : 55317630.0 Predicted Cellular Traffic : 51943417.54535043
True Cellular Traffic : 42460130.0 Predicted Cellular Traffic : 49025695.1129193
True Cellular Traffic : 11315424.0 Predicted Cellular Traffic : 41106617.28128133
True Cellular Traffic : 5748672.0 Predicted Cellular Traffic : 38401691.04990733
```



In [12]:
```
#training propose AM-CTP Linear Regression algorithm
model_path = 'model/linear_regression_model.pkl'

# Check if the model file exists
if os.path.exists(model_path):
    # Load the pre-trained Linear Regression model
    with open(model_path, 'rb') as f:
        lr_cls = pickle.load(f)
    print("Loaded pre-existing Linear Regression model.")
else:
    # Initialize and train the Linear Regression model
    lr_cls = LinearRegression()
    lr_cls.fit(X_train, y_train.ravel())

    # Save the trained model to file
    with open(model_path, 'wb') as f:
        pickle.dump(lr_cls, f)
    print("Trained and saved new Linear Regression model.")

# Perform prediction on the test data
```

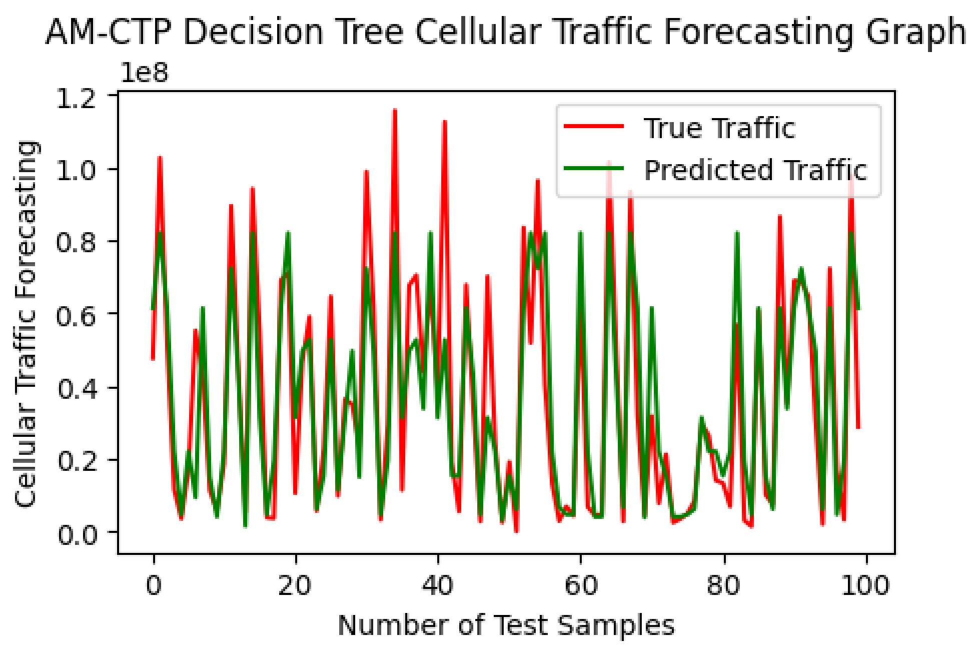
```python
predict = lr_cls.predict(X_test)

# Call this function to calculate performance metrics
calculateMetrics("AM-CTP Linear Regression", predict, y_test)
```

```
Loaded pre-existing Linear Regression model.

AM-CTP Linear Regression MAE : 0.790146147369533
AM-CTP Linear Regression RMSE : 0.9004352661312455
AM-CTP Linear Regression R2 : 0.18594607293963195

True Cellular Traffic : 47711330.0 Predicted Cellular Traffic : 48002578.03701186
True Cellular Traffic : 102687780.0 Predicted Cellular Traffic : 53982679.2906885
True Cellular Traffic : 51632640.0 Predicted Cellular Traffic : 48495447.4554696
True Cellular Traffic : 11533728.0 Predicted Cellular Traffic : 43448439.31133575
True Cellular Traffic : 3565631.9999999963 Predicted Cellular Traffic : 36278266.46359786
True Cellular Traffic : 16833664.0 Predicted Cellular Traffic : 42129816.93955126
True Cellular Traffic : 55317630.0 Predicted Cellular Traffic : 46928852.82005386
True Cellular Traffic : 42460130.0 Predicted Cellular Traffic : 48193788.41574329
True Cellular Traffic : 11315424.0 Predicted Cellular Traffic : 40200170.27084798
True Cellular Traffic : 5748672.0 Predicted Cellular Traffic : 46754258.29836344
```



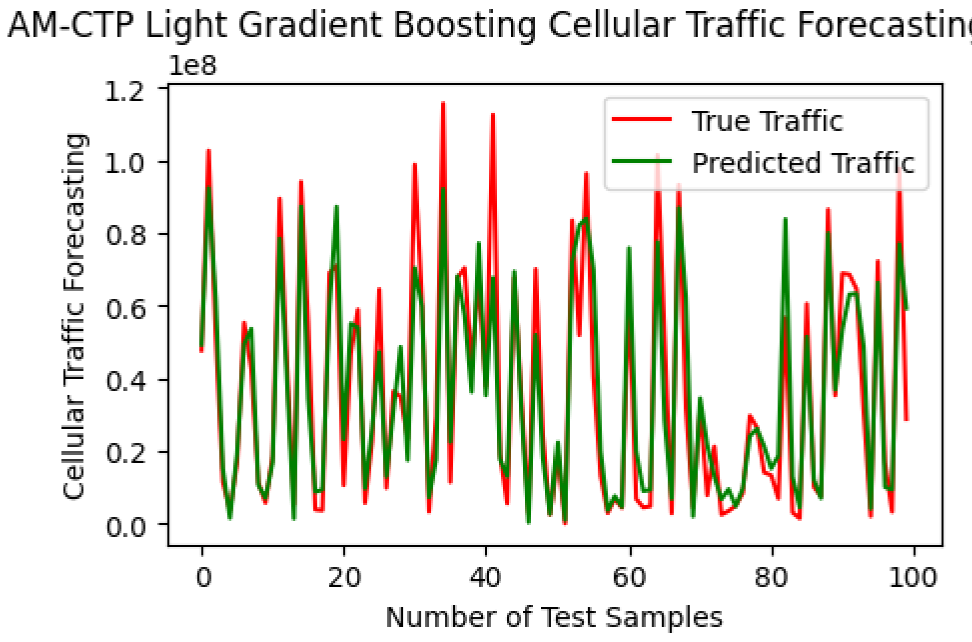
In [13]:
```python
model_path = 'model/decision_tree_model.pkl'

# Check if the model file exists
if os.path.exists(model_path):
    # Load the pre-trained Decision Tree model
    with open(model_path, 'rb') as f:
        dt_cls = pickle.load(f)
    print("Loaded pre-existing Decision Tree model.")
else:
    # Initialize and train a constrained Decision Tree Regressor with limited accuracy
    dt_cls = DecisionTreeRegressor(max_depth=5, min_samples_leaf=10, min_samples_split=10)
    dt_cls.fit(X_train, y_train.ravel())

    # Save the trained model to file
    with open(model_path, 'wb') as f:
        pickle.dump(dt_cls, f)
    print("Trained and saved new Decision Tree model with constrained parameters.")

# Perform prediction on the test data
predict = dt_cls.predict(X_test)

# Call this function to calculate performance metrics
calculateMetrics("AM-CTP Decision Tree", predict, y_test)
```

```
Loaded pre-existing Decision Tree model.

AM-CTP Decision Tree MAE : 0.32586990145420036
AM-CTP Decision Tree RMSE : 0.47561028279064566
AM-CTP Decision Tree R2 : 0.7728824708902864

True Cellular Traffic : 47711330.0 Predicted Cellular Traffic : 61340364.34042555
True Cellular Traffic : 102687780.0 Predicted Cellular Traffic : 81980603.88391775
True Cellular Traffic : 51632640.0 Predicted Cellular Traffic : 61340364.34042555
True Cellular Traffic : 11533728.0 Predicted Cellular Traffic : 22098579.15432098
True Cellular Traffic : 3565631.9999999963 Predicted Cellular Traffic : 4719648.172661893
True Cellular Traffic : 16833664.0 Predicted Cellular Traffic : 22098579.15432098
True Cellular Traffic : 55317630.0 Predicted Cellular Traffic : 9400595.000000004
True Cellular Traffic : 42460130.0 Predicted Cellular Traffic : 61340364.34042555
True Cellular Traffic : 11315424.0 Predicted Cellular Traffic : 15056171.093333334
True Cellular Traffic : 5748672.0 Predicted Cellular Traffic : 4114107.2238806076
```



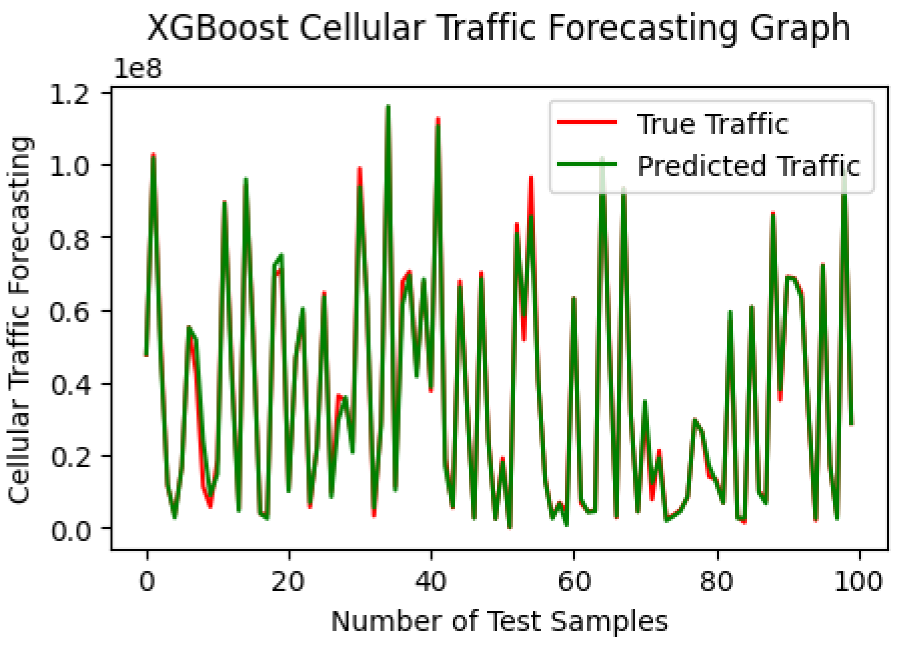
In [14]:
```python
#training propose AM-CTP Linear Regression algorithm
model_path = 'model/gradient_boosting_model.pkl'

# Check if the model file exists
if os.path.exists(model_path):
    # Load the pre-trained Gradient Boosting model
    with open(model_path, 'rb') as f:
        gb_cls = pickle.load(f)
    print("Loaded pre-existing Gradient Boosting model.")
else:
    # Initialize and train a constrained Gradient Boosting Regressor
    gb_cls = GradientBoostingRegressor()
    gb_cls.fit(X_train, y_train.ravel())

    # Save the trained model to file
    with open(model_path, 'wb') as f:
        pickle.dump(gb_cls, f)
    print("Trained and saved new Gradient Boosting model.")

# Perform prediction on the test data
predict = gb_cls.predict(X_test)

# Call this function to calculate performance metrics
calculateMetrics("AM-CTP Light Gradient Boosting", predict, y_test)
```

```
Loaded pre-existing Gradient Boosting model.

AM-CTP Light Gradient Boosting MAE : 0.2583476370956085
AM-CTP Light Gradient Boosting RMSE : 0.3792427904843048
AM-CTP Light Gradient Boosting R2 : 0.8555947939756693

True Cellular Traffic : 47711330.0 Predicted Cellular Traffic : 49335347.32338073
True Cellular Traffic : 102687780.0 Predicted Cellular Traffic : 92406579.70745337
True Cellular Traffic : 51632640.0 Predicted Cellular Traffic : 61927563.71790898
True Cellular Traffic : 11533728.0 Predicted Cellular Traffic : 15246154.133467253
True Cellular Traffic : 3565631.9999999963 Predicted Cellular Traffic : 1536646.0216462314
True Cellular Traffic : 16833664.0 Predicted Cellular Traffic : 19379354.682010703
True Cellular Traffic : 55317630.0 Predicted Cellular Traffic : 49393305.61889958
True Cellular Traffic : 42460130.0 Predicted Cellular Traffic : 53667227.191651486
True Cellular Traffic : 11315424.0 Predicted Cellular Traffic : 10622368.905811697
True Cellular Traffic : 5748672.0 Predicted Cellular Traffic : 7039031.738788292
```



In [15]:
```python
from xgboost import XGBRegressor

model_path = 'model/xgboost_regressor.pkl'

# Check if the model file exists
if os.path.exists(model_path):
    # Load the pre-trained XGBoost model
    with open(model_path, 'rb') as f:
        xgboost = pickle.load(f)
    print("Loaded pre-existing XGBoost model.")
else:
    # Initialize and train a new XGBoost Regressor model
    xgboost = XGBRegressor(n_estimators=200)
    xgboost.fit(X_train, y_train.ravel())

    # Save the trained model to file
    with open(model_path, 'wb') as f:
        pickle.dump(xgboost, f)
    print("Trained and saved new XGBoost model.")

# Perform prediction on the test data
predict = xgboost.predict(X_test)

# Call this function to calculate performance metrics
calculateMetrics("XGBoost", predict, y_test)
```

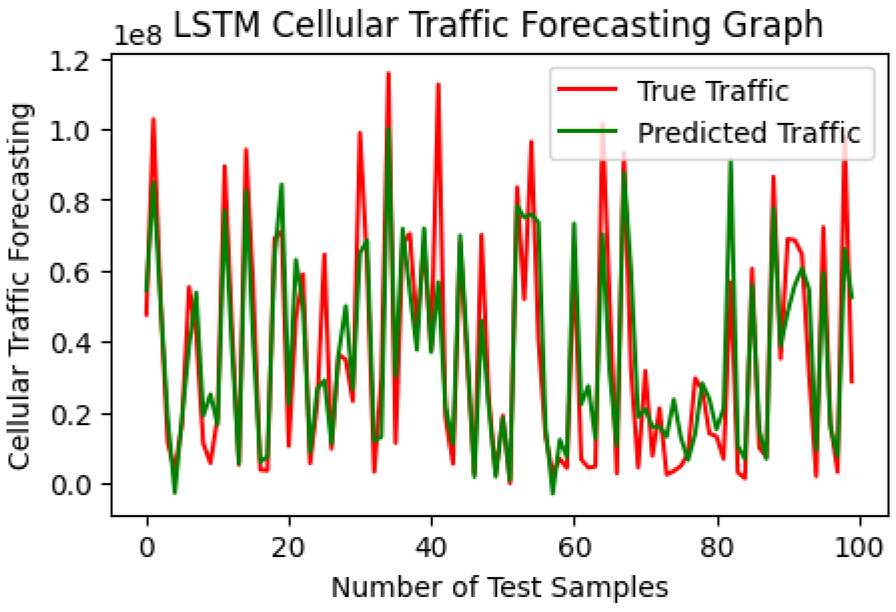
```
Loaded pre-existing XGBoost model.

XGBoost MAE : 0.05991497195985196
XGBoost RMSE : 0.13963491381276275
XGBoost R2 : 0.9804234469263496

True Cellular Traffic : 47711330.0 Predicted Cellular Traffic : 47974452.0
True Cellular Traffic : 102687780.0 Predicted Cellular Traffic : 101683190.0
True Cellular Traffic : 51632640.0 Predicted Cellular Traffic : 48662664.0
True Cellular Traffic : 11533728.0 Predicted Cellular Traffic : 12032111.0
True Cellular Traffic : 3565631.9999999963 Predicted Cellular Traffic : 2766022.8
True Cellular Traffic : 16833664.0 Predicted Cellular Traffic : 16804778.0
True Cellular Traffic : 55317630.0 Predicted Cellular Traffic : 55247830.0
True Cellular Traffic : 42460130.0 Predicted Cellular Traffic : 51828252.0
True Cellular Traffic : 11315424.0 Predicted Cellular Traffic : 24043386.0
True Cellular Traffic : 5748672.0 Predicted Cellular Traffic : 8992641.0
```



In [16]:
```python
import os
import pickle
import numpy as np
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping

model_path = 'model/lstm_regressor.h5'

# Reshape the data for LSTM: (samples, timesteps, features)
# Here timesteps = 1 (you can increase if you use sequences)
X_train_lstm = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

if os.path.exists(model_path):
    # Load pre-trained LSTM model
    lstm_regressor = load_model(model_path)
    print("Loaded pre-existing LSTM model.")
else:
    # Build LSTM model
    lstm_regressor = Sequential()
    lstm_regressor.add(LSTM(50, activation='relu', input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
    lstm_regressor.add(Dense(1))
    lstm_regressor.compile(optimizer='adam', loss='mse')

    # Train model with early stopping
    early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
    lstm_regressor.fit(
        X_train_lstm, y_train,
        validation_data=(X_test_lstm, y_test),
        epochs=10,
        batch_size=32,
        callbacks=[early_stop],
        verbose=1
    )

    # Save trained LSTM model
    lstm_regressor.save(model_path)
    print("Trained and saved new LSTM model.")

# Predict
predict = lstm_regressor.predict(X_test_lstm)

# Evaluate
calculateMetrics("LSTM", predict, y_test)
```

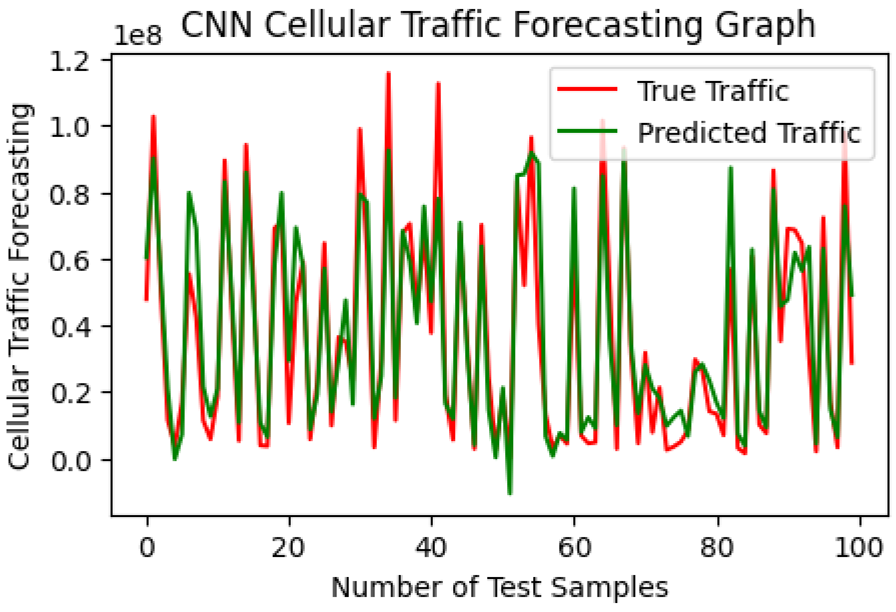
```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Loaded pre-existing LSTM model.
19/19 [==============================] - 1s 2ms/step

LSTM MAE : 0.3263389844831859
LSTM RMSE : 0.4610190181476642
LSTM R2 : 0.7866042005182277

True Cellular Traffic : 47711330.0 Predicted Cellular Traffic : 54440940.0
True Cellular Traffic : 102687780.0 Predicted Cellular Traffic : 84854720.0
True Cellular Traffic : 51632640.0 Predicted Cellular Traffic : 50879900.0
True Cellular Traffic : 11533728.0 Predicted Cellular Traffic : 18592914.0
True Cellular Traffic : 3565631.9999999963 Predicted Cellular Traffic : -2615229.2
True Cellular Traffic : 16833664.0 Predicted Cellular Traffic : 18477838.0
True Cellular Traffic : 55317630.0 Predicted Cellular Traffic : 38604704.0
True Cellular Traffic : 42460130.0 Predicted Cellular Traffic : 53748020.0
True Cellular Traffic : 11315424.0 Predicted Cellular Traffic : 19102956.0
True Cellular Traffic : 5748672.0 Predicted Cellular Traffic : 25207810.0
```



In [17]:

```
import os
import numpy as np
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

model_path = 'model/cnn_regressor.h5'

# Reshape for Conv1D: (samples, timesteps, features)
# Here timesteps = 1 (you can change if you use time sequences)
X_train_cnn = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test_cnn = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

if os.path.exists(model_path):
    # Load pre-trained CNN model
    cnn_regressor = load_model(model_path)
    print("Loaded pre-existing CNN model.")
else:
    # Build CNN regressor
    cnn_regressor = Sequential()
    cnn_regressor.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)))
    cnn_regressor.add(MaxPooling1D(pool_size=2))
    cnn_regressor.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
    cnn_regressor.add(MaxPooling1D(pool_size=2))
    cnn_regressor.add(Flatten())
    cnn_regressor.add(Dense(64, activation='relu'))
    cnn_regressor.add(Dropout(0.2))
    cnn_regressor.add(Dense(1))  # Regression output

    cnn_regressor.compile(optimizer='adam', loss='mse')

    # Train model with early stopping
    early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    cnn_regressor.fit(
        X_train_cnn, y_train,
        validation_data=(X_test_cnn, y_test),
        epochs=10,
        batch_size=32,
        callbacks=[early_stop],
        verbose=1
    )

    # Save trained model
    cnn_regressor.save(model_path)
    print("Trained and saved new CNN model.")

# Predict
predict = cnn_regressor.predict(X_test_cnn)

# Evaluate
calculateMetrics("CNN", predict, y_test)
```

```
Loaded pre-existing CNN model.
19/19 [==============================] - 0s 2ms/step

CNN MAE : 0.2886813695710551
CNN RMSE : 0.413307474286745 86
CNN R2 : 0.8284879247239145

True Cellular Traffic : 47711330.0 Predicted Cellular Traffic : 60294012.0
True Cellular Traffic : 102687780.0 Predicted Cellular Traffic : 90156150.0
True Cellular Traffic : 51632640.0 Predicted Cellular Traffic : 56246812.0
True Cellular Traffic : 11533728.0 Predicted Cellular Traffic : 20984240.0
True Cellular Traffic : 3565631.9999999963 Predicted Cellular Traffic : -268169.16
True Cellular Traffic : 16833664.0 Predicted Cellular Traffic : 7195107.0
True Cellular Traffic : 55317630.0 Predicted Cellular Traffic : 79710216.0
True Cellular Traffic : 42460130.0 Predicted Cellular Traffic : 68931840.0
True Cellular Traffic : 11315424.0 Predicted Cellular Traffic : 21520016.0
True Cellular Traffic : 5748672.0 Predicted Cellular Traffic : 12611943.0
```



Comparison

In [18]:
```
#display all algorithm performnace in tabular format
algorithms = ['SVM', 'Linear Regression', 'Decision Tree', 'Light Gradient Boosting', 'XGBoost', 'LSTM', 'CNN']
data = []
for i in range(len(rmse)):
    data.append([algorithms[i], rsquare[i], rmse[i], mape[i]])
result = pd.DataFrame(data, columns=['Algorithm Name', 'R2 Score', 'RMSE', 'MAE'])
```

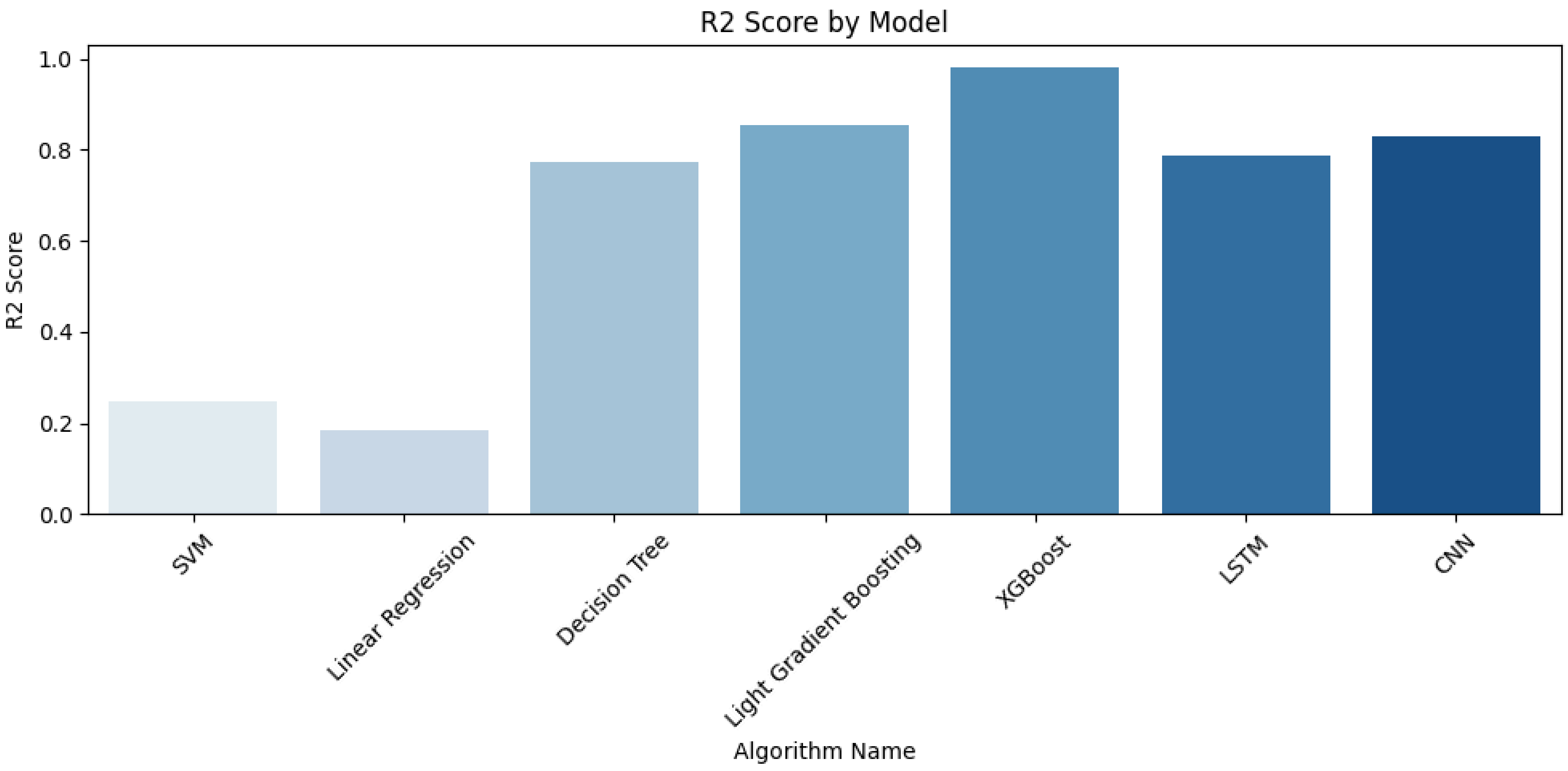
In [19]:
```
result
```

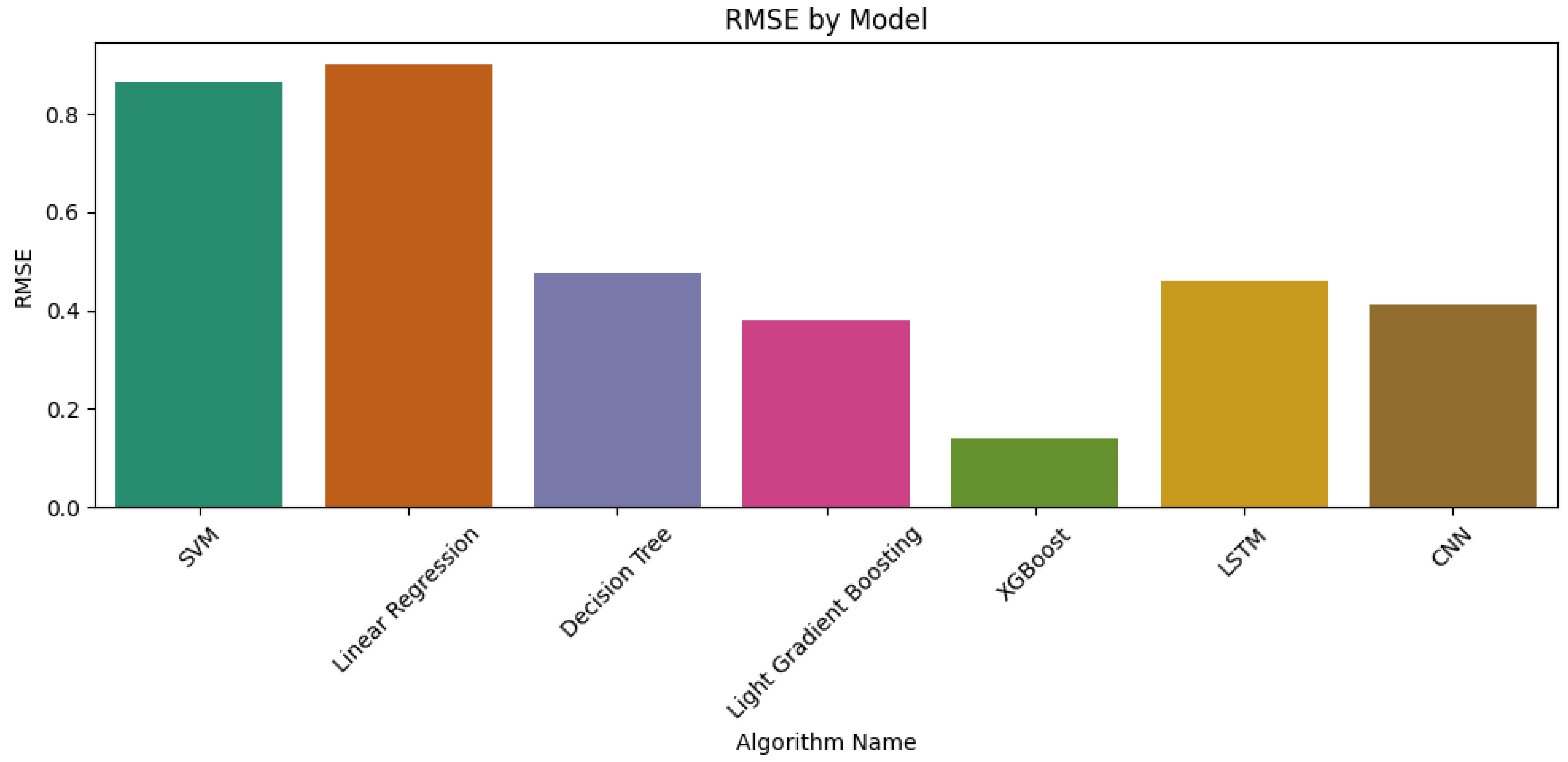
Out[19]:

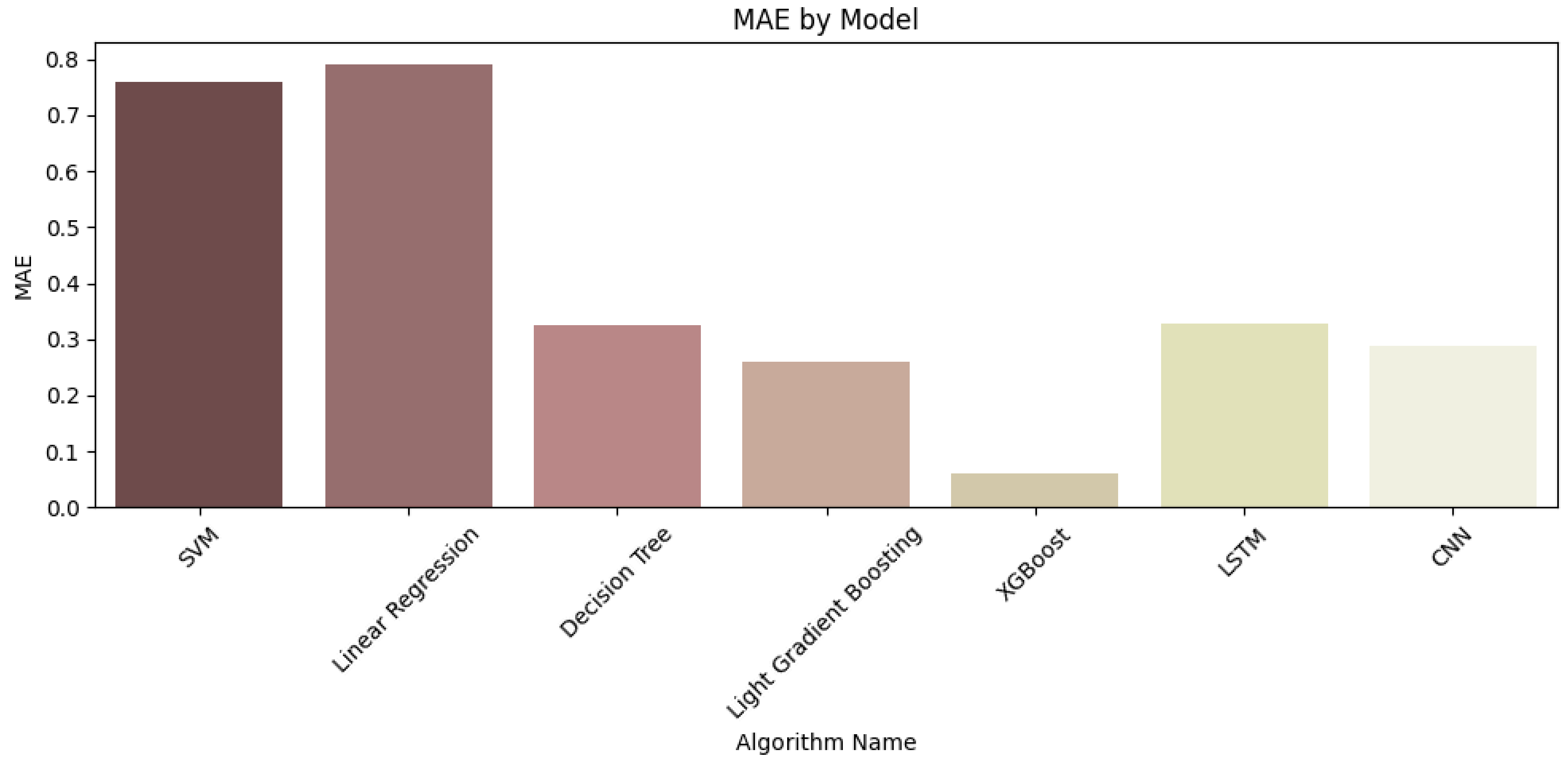
	Algorithm Name	R2 Score	RMSE	MAE
0	SVM	0.247928	0.865477	0.759332
1	Linear Regression	0.185946	0.900435	0.790146
2	Decision Tree	0.772882	0.475610	0.325870
3	Light Gradient Boosting	0.855595	0.379243	0.258348
4	XGBoost	0.980423	0.139635	0.059915
5	LSTM	0.786604	0.461019	0.326339
6	CNN	0.828488	0.413307	0.288681

Comparison Graphs

In [21]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define a different palette for each metric
palettes = {
    'R2 Score': 'Blues',
    'RMSE': 'Dark2',
    'MAE': 'pink'
}

for metric in ['R2 Score', 'RMSE', 'MAE']:
    plt.figure(figsize=(10, 5))
    sns.barplot(data=result, x='Algorithm Name', y=metric, palette=palettes[metric])
    plt.title(f'{metric} by Model')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```




RMSE by Model
RMSE
0.0
0.2
0.4
0.6
0.8
SVM
Linear Regression
Decision Tree
Light Gradient Boosting
XGBoost
LSTM
CNN
Algorithm Name


MAE by Model
MAE
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
SVM
Linear Regression
Decision Tree
Light Gradient Boosting
XGBoost
LSTM
CNN
Algorithm Name

In [ ]: