Resource Allocation in 5G Network Service

In [191]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [192]:

```
df = pd.read_csv('/kaggle/input/5g-quality-of-service/Quality of Service 5G.csv')
```

In [193]:

```
df
```

Out[193]:

	Timestamp	User_ID	Application_Type	Signal_Strength	Latency	Required_Band
0	9/3/2023 10:00	User_1	Video_Call	-75 dBm	30 ms	10
1	9/3/2023 10:00	User_2	Voice_Call	-80 dBm	20 ms	100
2	9/3/2023 10:00	User_3	Streaming	-85 dBm	40 ms	5
3	9/3/2023 10:00	User_4	Emergency_Service	-70 dBm	10 ms	1
4	9/3/2023 10:00	User_5	Online_Gaming	-78 dBm	25 ms	2
...	...	...	...	...	...	
395	9/3/2023 10:06	User_396	Streaming	-110 dBm	61 ms	1.3
396	9/3/2023 10:06	User_397	Video_Call	-40 dBm	53 ms	14.5
397	9/3/2023 10:06	User_398	Video_Streaming	-113 dBm	58 ms	1.0
398	9/3/2023 10:06	User_399	Emergency_Service	-40 dBm	5 ms	0.4
399	9/3/2023 10:06	User_400	Web_Browsing	-113 dBm	0 ms	0.1

400 rows × 8 columns

1. Data Pre-Processing :

In [194]:

```
# Get all insights from the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Timestamp            400 non-null    object
 1   User_ID              400 non-null    object
 2   Application_Type     400 non-null    object
 3   Signal_Strength      400 non-null    object
 4   Latency              400 non-null    object
 5   Required_Bandwidth   400 non-null    object
 6   Allocated_Bandwidth  400 non-null    object
 7   Resource_Allocation  400 non-null    object
dtypes: object(8)
memory usage: 25.1+ KB
```

In [195]:

```
# Check for NULL Values
df.isna().sum()
```

Out[195]:

```
Timestamp              0
User_ID                0
Application_Type       0
Signal_Strength        0
Latency                0
Required_Bandwidth     0
Allocated_Bandwidth    0
Resource_Allocation    0
dtype: int64
```

In [196]:

```
# Check for Duplicates
df.duplicated().sum()
```

Out[196]:

```
0
```

In [197]:

```
# Start with encoding df['Application_Type']

df['Application_Type'].unique()
```

Out[197]:

```
array(['Video_Call', 'Voice_Call', 'Streaming', 'Emergency_Service',
       'Online_Gaming', 'Background_Download', 'Web_Browsing',
       'IoT_Temperature', 'Video_Streaming', 'File_Download', 'VoIP_Ca
ll'],
      dtype=object)
```

df['Application_Type']

- Contains the following unique values: ['Video_Call', 'Voice_Call', 'Streaming', 'Emergency_Service', 'Online_Gaming', 'Background_Download', 'Web_Browsing', 'IoT_Temperature', 'Video_Streaming', 'File_Download', 'VoIP_Call']
- These all categories can be arranged in increasing Ordered form of Resource requirement:

  1. IoT_Temperature
  2. Web_Browsing
  3. VoIP_Call
  4. Video_Call
  5. File_Download
  6. Background_Download
  7. Streaming
  8. Video_Streaming
  9. Online_Gaming
  10. Emergency_Service
- So, use Ordinal Encoding technique to transform the text data to numeric values.

In [198]:

```
from sklearn.preprocessing import OrdinalEncoder

# Arrange data in ordered categories
data = [['IoT_Temperature', 'Web_Browsing', 'VoIP_Call', 'Voice_Call', 'Video_Cal
l', 'File_Download',
    'Background_Download', 'Streaming', 'Video_Streaming', 'Online_Gaming', 'Emerge
ncy_Service']]

encoder = OrdinalEncoder(categories=data)

print(df['Application_Type'].unique())
df['Application_Type'] = encoder.fit_transform(np.array(df['Application_Type']).res
hape(-1,1))
print(df['Application_Type'].unique())
```

```
['Video_Call' 'Voice_Call' 'Streaming' 'Emergency_Service' 'Online_Gam
ing'
 'Background_Download' 'Web_Browsing' 'IoT_Temperature' 'Video_Streami
ng'
 'File_Download' 'VoIP_Call']
[ 4.  3.  7. 10.  9.  6.  1.  0.  8.  5.  2.]
```

In [199]:

```
# Remove unwanted substr and convert df['Resource_Allocation'] to int dtype

df['Resource_Allocation'] = df['Resource_Allocation'].str.replace('%','').astype('i
nt')
```

In [200]:

```
# Remove unwanted substr and convert df['User_ID'] to int dtype

df['User_ID'] = df['User_ID'].str.replace('User_','').astype('int')
```

In [201]:

```
# Bandwidth contains 2 units : Kbps and Mbps, where Kbps=1000*Mbps
# Mbps : Convert to Kbps
# Kbps : Leave as it is

def mbps_to_kbps(value):
    if 'Mbps' in value:
        n = float(value.replace(' Mbps',''))
        return str(n*1000)+' Kbps'
    else:
        return value


df['Required_Bandwidth'] = df['Required_Bandwidth'].map(mbps_to_kbps)
df['Allocated_Bandwidth'] = df['Allocated_Bandwidth'].map(mbps_to_kbps)
```

In [202]:

```
# Remove unwanted substr and convert df['Required_Bandwidth'] and
# df['Allocated_Bandwidth'] to int dtype

df['Required_Bandwidth'] = df['Required_Bandwidth'].str.replace(' Kbps','').astype
('float')
df['Allocated_Bandwidth'] = df['Allocated_Bandwidth'].str.replace(' Kbps','').astyp
e('float')
```

In [203]:

```
# Remove unwanted substr and convert df['Latency'] to int dtype
df['Latency'] = df['Latency'].str.replace(' ms','').astype('int')
```

In [204]:

```
# Remove unwanted substr and convert df['Signal_Strength'] to int dtype
df['Signal_Strength'] = df['Signal_Strength'].str.replace(' dBm','').astype('int')
```

In [205]:

```
# Cast 'Timestamp' col dtype from object to datetime
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
```

In [206]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Timestamp            400 non-null    datetime64[ns]
 1   User_ID              400 non-null    int64
 2   Application_Type     400 non-null    float64
 3   Signal_Strength      400 non-null    int64
 4   Latency              400 non-null    int64
 5   Required_Bandwidth   400 non-null    float64
 6   Allocated_Bandwidth  400 non-null    float64
 7   Resource_Allocation  400 non-null    int64
dtypes: datetime64[ns](1), float64(3), int64(4)
memory usage: 25.1 KB
```

In [207]:

```
df.head()
```

Out[207]:

	Timestamp	User_ID	Application_Type	Signal_Strength	Latency	Required_Bandwidth
0	2023-09-03 10:00:00	1	4.0	-75	30	10000.0
1	2023-09-03 10:00:00	2	3.0	-80	20	100.0
2	2023-09-03 10:00:00	3	7.0	-85	40	5000.0
3	2023-09-03 10:00:00	4	10.0	-70	10	1000.0
4	2023-09-03 10:00:00	5	9.0	-78	25	2000.0

In [208]:

```
df.tail()
```

Out[208]:

	Timestamp	User_ID	Application_Type	Signal_Strength	Latency	Required_Bandwic
395	2023-09-03 10:06:00	396	7.0	-110	61	130(
396	2023-09-03 10:06:00	397	4.0	-40	53	1450(
397	2023-09-03 10:06:00	398	8.0	-113	58	100(
398	2023-09-03 10:06:00	399	10.0	-40	5	40(
399	2023-09-03 10:06:00	400	1.0	-113	0	10(

In [209]:

```
df.describe()
```

Out[209]:

	Timestamp	User_ID	Application_Type	Signal_Strength	Latency	Required_
count	400	400.000000	400.000000	400.000000	400.000000	
mean	2023-09-03 10:03:00	200.500000	5.605000	-80.495000	33.825000	3
min	2023-09-03 10:00:00	1.000000	0.000000	-123.000000	0.000000	
25%	2023-09-03 10:01:00	100.750000	2.000000	-98.000000	21.750000	
50%	2023-09-03 10:03:00	200.500000	6.000000	-83.000000	31.000000	1
75%	2023-09-03 10:05:00	300.250000	8.000000	-64.000000	45.000000	4
max	2023-09-03 10:06:00	400.000000	10.000000	-40.000000	110.000000	14
std	NaN	115.614301	3.156562	20.701119	21.122139	3

In [210]:

```
# 'User_ID' having very less effect on result, so remove it
df.drop('User_ID', axis=1, inplace=True)
```

In [211]:

```
df
```

Out[211]:

	Timestamp	Application_Type	Signal_Strength	Latency	Required_Bandwidth	Alloc
0	2023-09-03 10:00:00	4.0	-75	30	10000.0	
1	2023-09-03 10:00:00	3.0	-80	20	100.0	
2	2023-09-03 10:00:00	7.0	-85	40	5000.0	
3	2023-09-03 10:00:00	10.0	-70	10	1000.0	
4	2023-09-03 10:00:00	9.0	-78	25	2000.0	
...	...	...	...	...	...	
395	2023-09-03 10:06:00	7.0	-110	61	1300.0	
396	2023-09-03 10:06:00	4.0	-40	53	14500.0	
397	2023-09-03 10:06:00	8.0	-113	58	1000.0	
398	2023-09-03 10:06:00	10.0	-40	5	400.0	
399	2023-09-03 10:06:00	1.0	-113	0	100.0	

400 rows × 7 columns

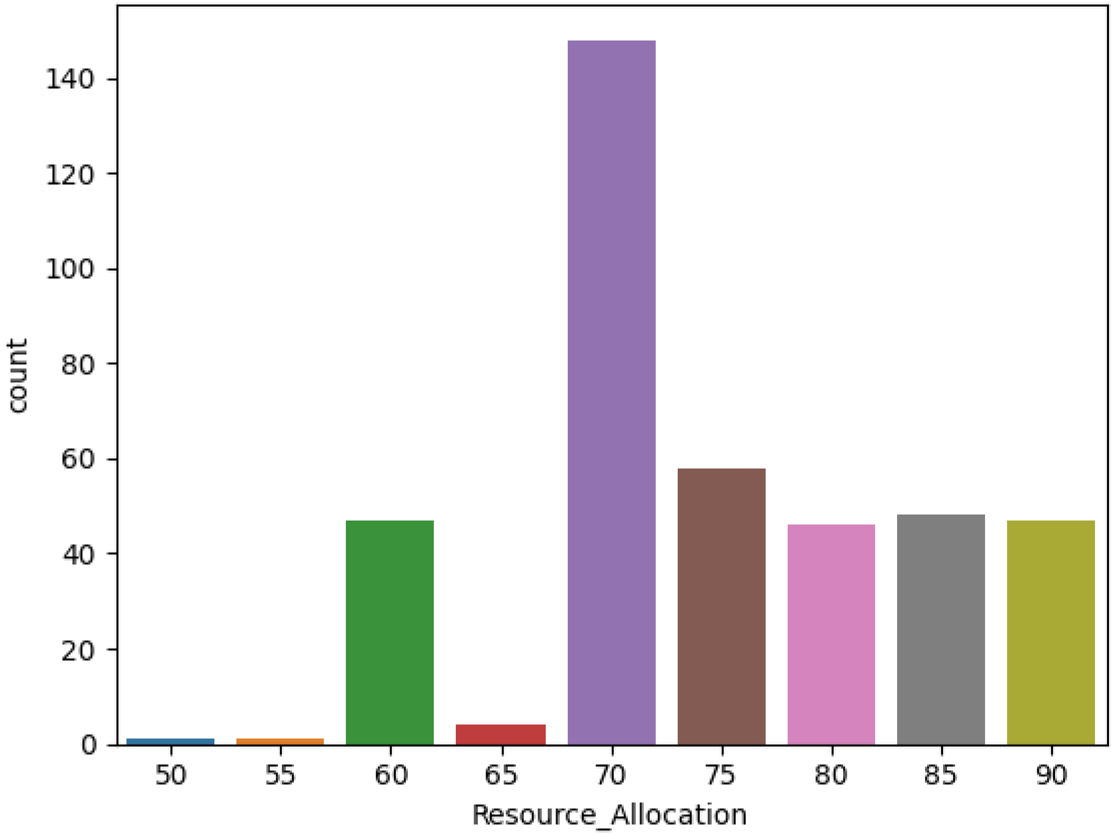
Data Visualization :

In [212]:

```
# Find count of values in target variable
sns.countplot(x=df['Resource_Allocation'])
```

Out[212]:

```
<Axes: xlabel='Resource_Allocation', ylabel='count'>
```

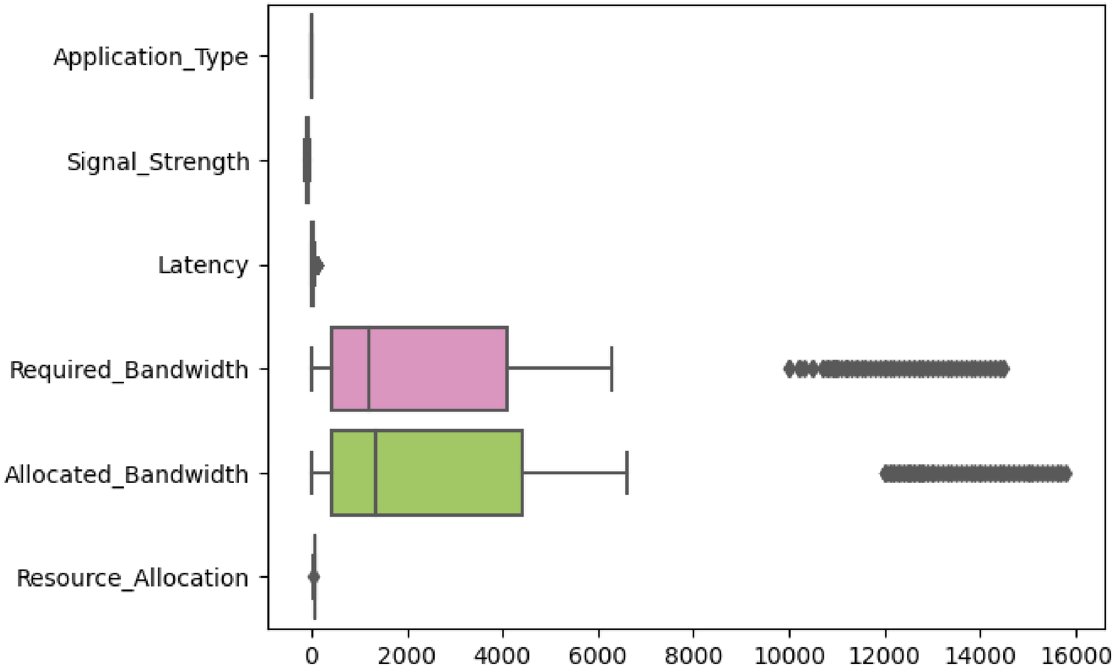


In [213]:

```
# Finding Outliers

sns.boxplot(data=df, orient="h", palette="Set2")  # Use sns.violinplot() for a viol
in plot
```

Out[213]:

<Axes: >

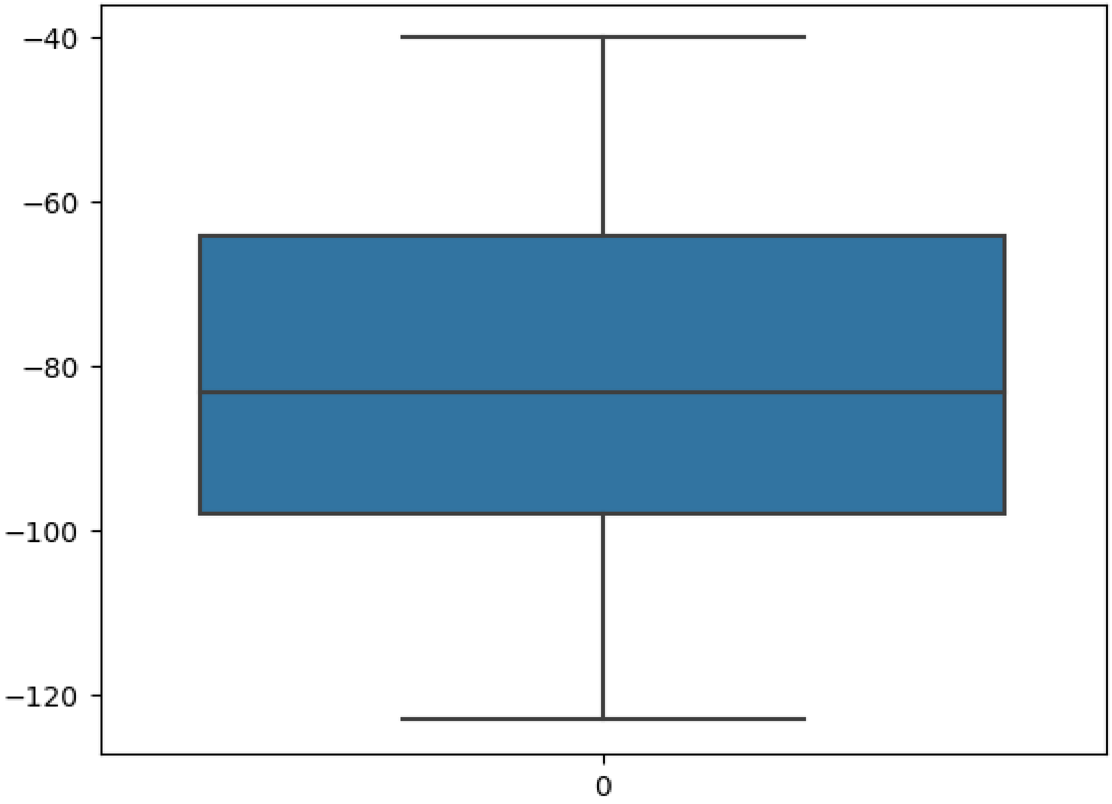


In [214]:

```
sns.boxplot(data=df['Signal_Strength'])
# No outtliers present
```

Out[214]:

<Axes: >

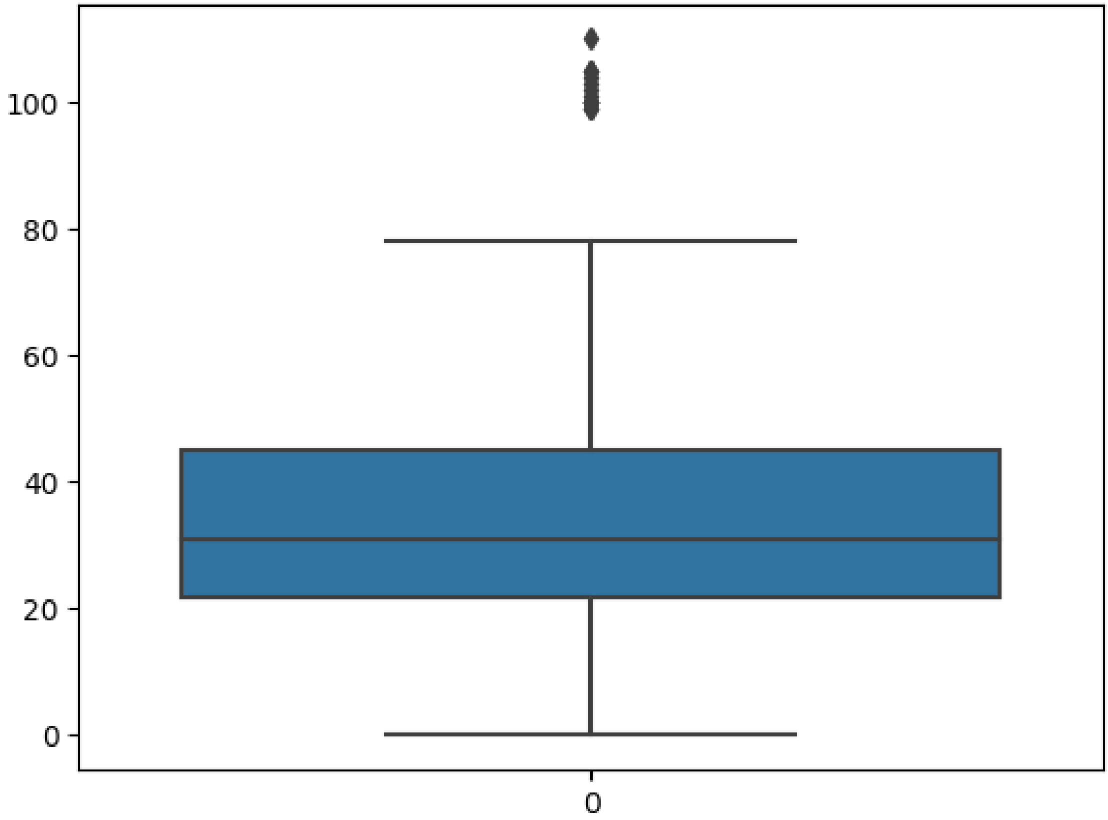


In [215]:

```
sns.boxplot(data=df['Latency'])
# Outliers present on positive side
```

Out[215]:

<Axes: >



In [216]:

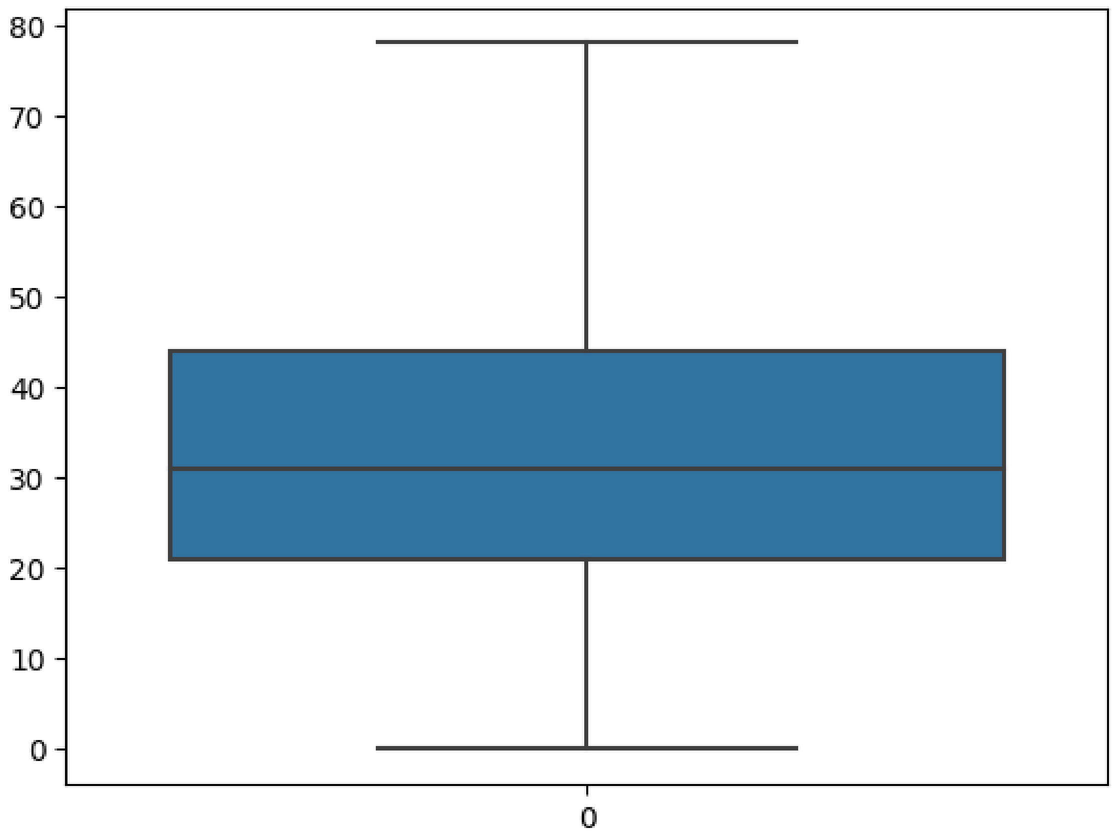
```
# Cannot transform latency data to handle outliers=> Leads to greater number of out
liers
# So, Removing outliers from 'Latency'

Q1 = np.percentile(df['Latency'], 25, method='midpoint')
Q3 = np.percentile(df['Latency'], 75, method='midpoint')

IQR = Q3-Q1

upper = Q3 + 1.5*IQR
lower = Q1 - 1.5*IQR

upper_val = np.where(df['Latency'] >= upper)
lower_val = np.where(df['Latency'] <= lower)

# Removing the outliers
df.drop(upper_val[0], inplace=True)
df.drop(lower_val[0], inplace=True)
```

In [217]:

```
sns.boxplot(data=df['Latency'])
```

Out[217]:

<Axes: >



In [218]:

```
df.shape
```

Out[218]:

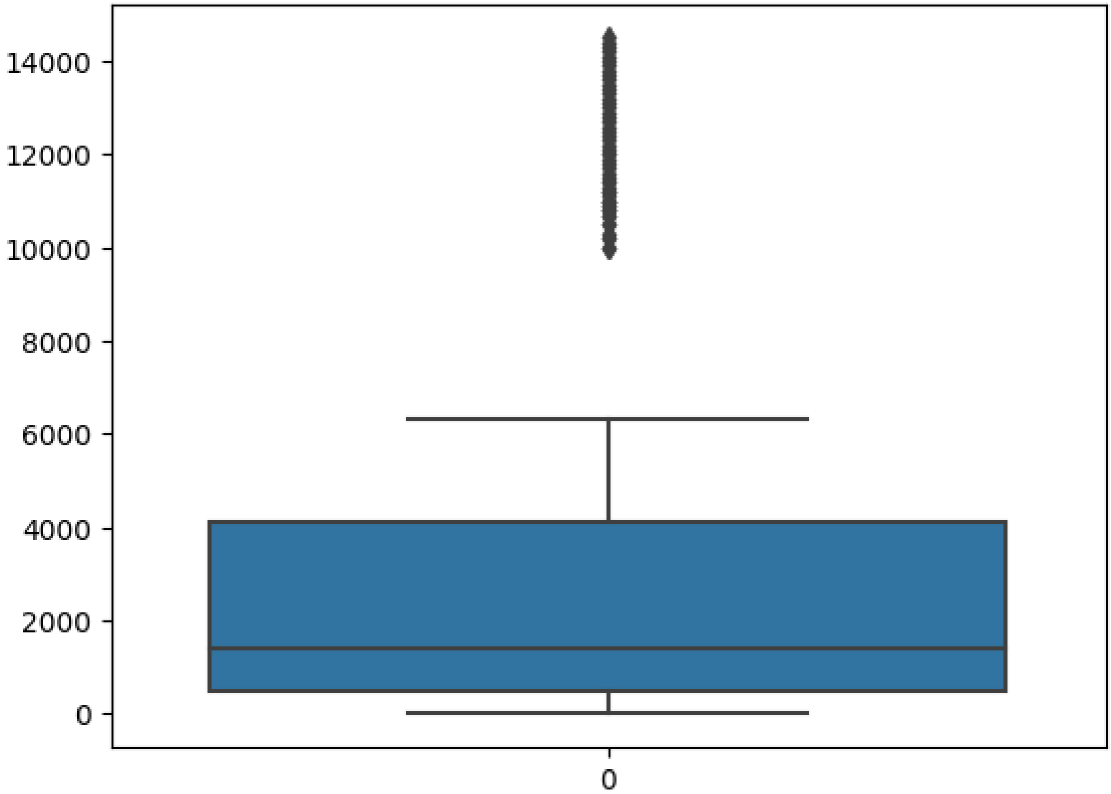
(387, 7)

In [219]:

```
sns.boxplot(data=df['Required_Bandwidth'])
# Outliers present on positive side
```

Out[219]:

<Axes: >

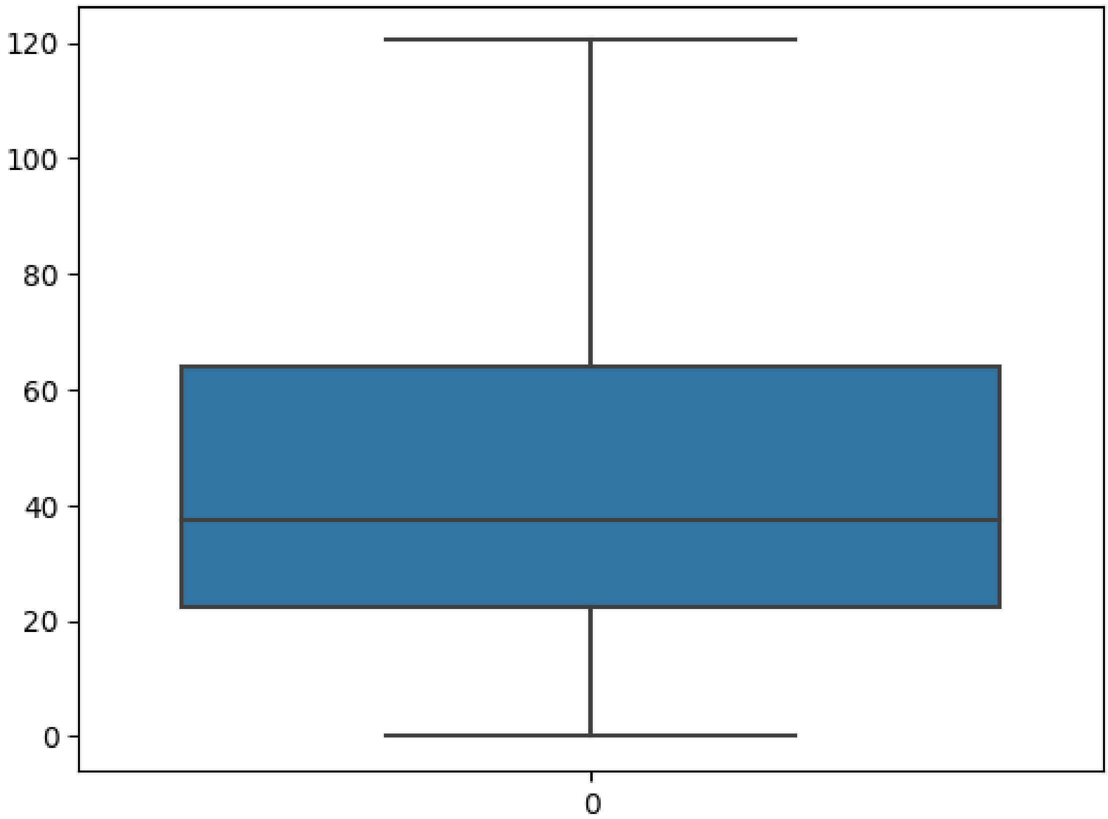


In [220]:

```
# Can handle outliers in 'Required_Bandwidth', using Transformation
# So, use "Square Root Transformation (np.sqrt())""

df['Required_Bandwidth'] = np.sqrt(df['Required_Bandwidth'])

sns.boxplot(data=df['Required_Bandwidth'])
```

Out[220]:

```
<Axes: >
```

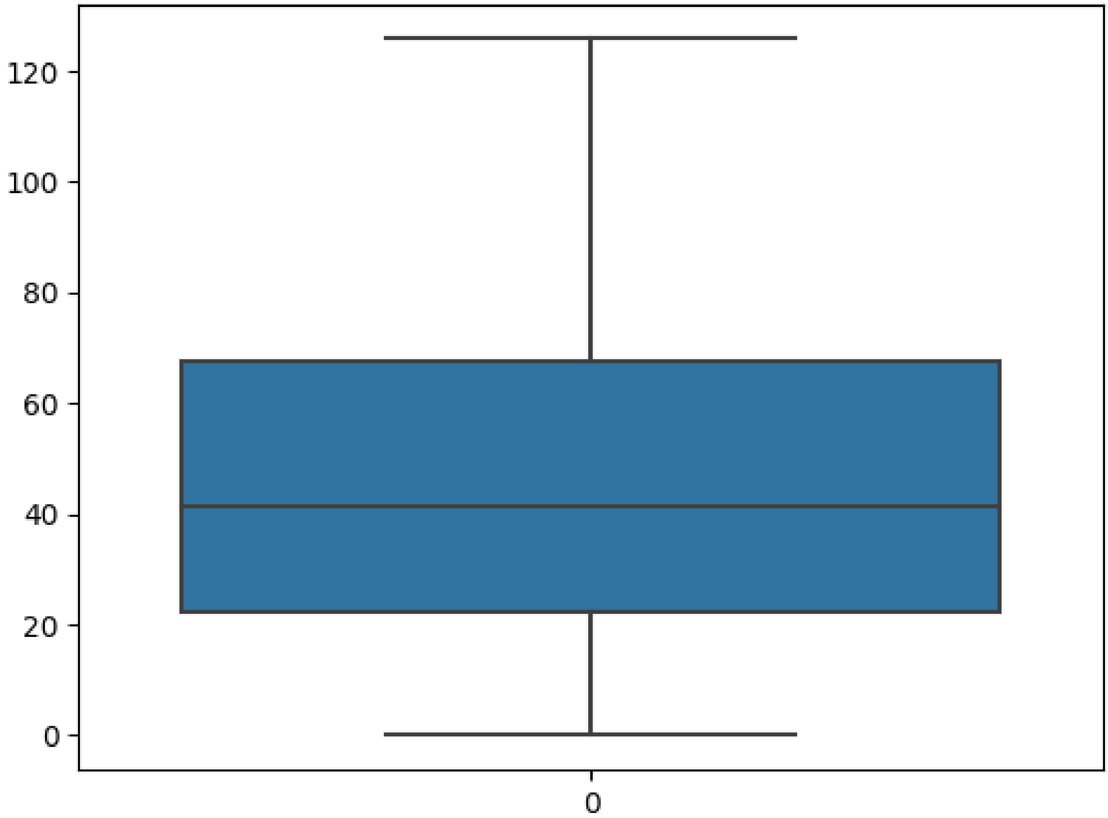


In [221]:

```
df['Allocated_Bandwidth'] = np.sqrt(df['Allocated_Bandwidth'])

sns.boxplot(data=df['Allocated_Bandwidth'])
```

Out[221]:

<Axes: >

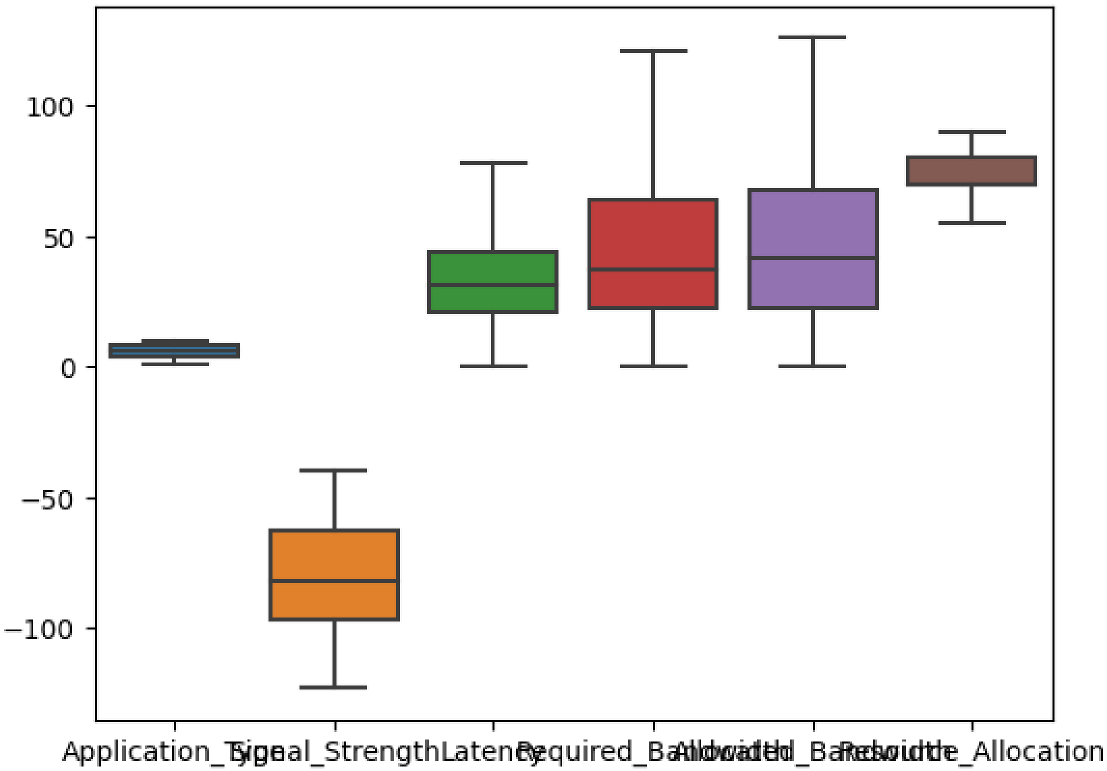


In [222]:

```
sns.boxplot(data=df)
# Outliers removed from all the columns
# But data is present over very wide ranges
# So, have to normalize data to one range
```

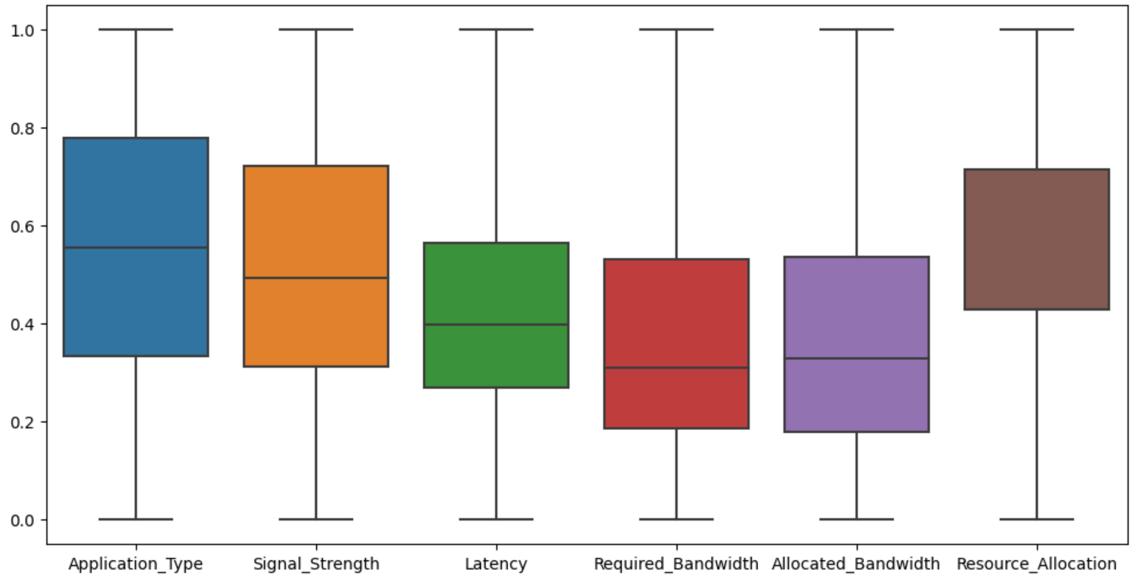
Out[222]:

<Axes: >



In [223]:

```
# Define the min-max scaling function
def min_max_scaling(column):
    min_val = column.min()
    max_val = column.max()
    return (column - min_val) / (max_val - min_val)

Transformed_df = df

Transformed_df['Signal_Strength'] = min_max_scaling(df['Signal_Strength'])
Transformed_df['Latency'] = min_max_scaling(df['Latency'])
Transformed_df['Required_Bandwidth'] = min_max_scaling(df['Required_Bandwidth'])
Transformed_df['Allocated_Bandwidth'] = min_max_scaling(df['Allocated_Bandwidth'])
Transformed_df['Resource_Allocation'] = min_max_scaling(df['Resource_Allocation'])
Transformed_df['Application_Type'] = min_max_scaling(df['Application_Type'])

Transformed_df.drop('Timestamp', axis=1, inplace=True)

plt.figure(figsize=(12,6))
sns.boxplot(data=Transformed_df)
plt.show()

# Thus, all columns data now comes in same range
```



In [224]:

```
Transformed_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 387 entries, 0 to 399
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Application_Type     387 non-null    float64
 1   Signal_Strength      387 non-null    float64
 2   Latency              387 non-null    float64
 3   Required_Bandwidth   387 non-null    float64
 4   Allocated_Bandwidth  387 non-null    float64
 5   Resource_Allocation  387 non-null    float64
dtypes: float64(6)
memory usage: 29.3 KB
```

In [225]:

```
Transformed_df.describe()
```

Out[225]:

	Application_Type	Signal_Strength	Latency	Required_Bandwidth	Allocated_Ba
count	387.000000	387.000000	387.000000	387.000000	387
mean	0.532587	0.520096	0.404227	0.386077	(
std	0.337124	0.249585	0.221440	0.273194	(
min	0.000000	0.000000	0.000000	0.000000	(
25%	0.333333	0.313253	0.269231	0.185695	(
50%	0.555556	0.493976	0.397436	0.310728	(
75%	0.777778	0.722892	0.564103	0.531751	(
max	1.000000	1.000000	1.000000	1.000000	'

Split the DataFrame :

In [226]:

```
X = Transformed_df.drop('Resource_Allocation', axis=1)
Y = Transformed_df['Resource_Allocation']
```

In [227]:

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(X, Y, test_size=0.20, random_state
=42)
```

In [228]:

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[228]:

```
((309, 5), (78, 5), (309,), (78,))
```

Linear Regression Model :

Create and Train the Model :

In [229]:

```
from sklearn.linear_model import LinearRegression
```

In [230]:

```
lin_regressor = LinearRegression()
lin_regressor.fit(x_train,y_train)
```

Out[230]:

```
▾ LinearRegression
LinearRegression()
```

Predict Test Set Results :

In [231]:

```
lin_regressor_y_pred = lin_regressor.predict(x_test)
```

Evaluation of Model Performance :

In [232]:

```
from sklearn.metrics import r2_score, mean_squared_error
```

In [233]:

```
lin_regressor_r2 = r2_score(y_test, lin_regressor_y_pred)
lin_regressor_r2
```

Out[233]:

```
-0.048720141610214984
```

In [234]:

```
lin_regressor_mse = mean_squared_error(y_test, lin_regressor_y_pred)
lin_regressor_mse
```

Out[234]:

```
0.07650919226033008
```

In [235]:

```
lin_regressor_rmse = np.sqrt(lin_regressor_mse)
lin_regressor_rmse
```

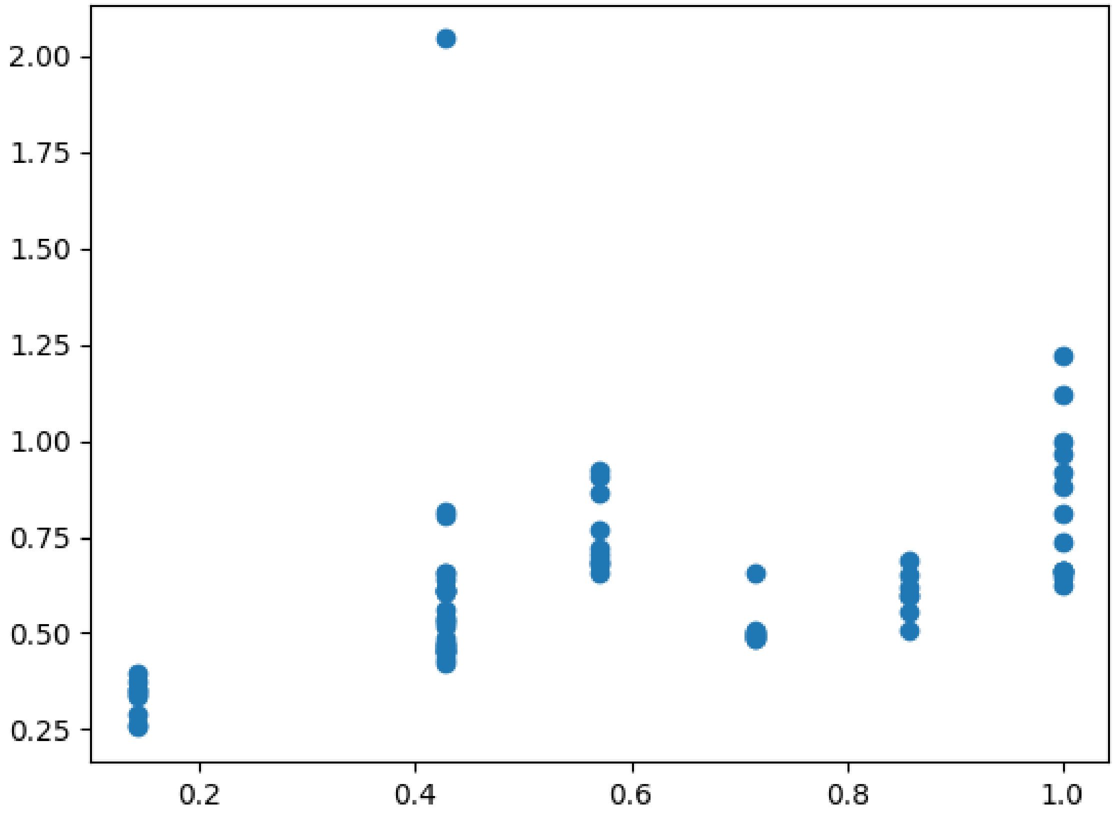
Out[235]:

```
0.2766029505633121
```

Visualize the Results of the Prediction :

In [236]:

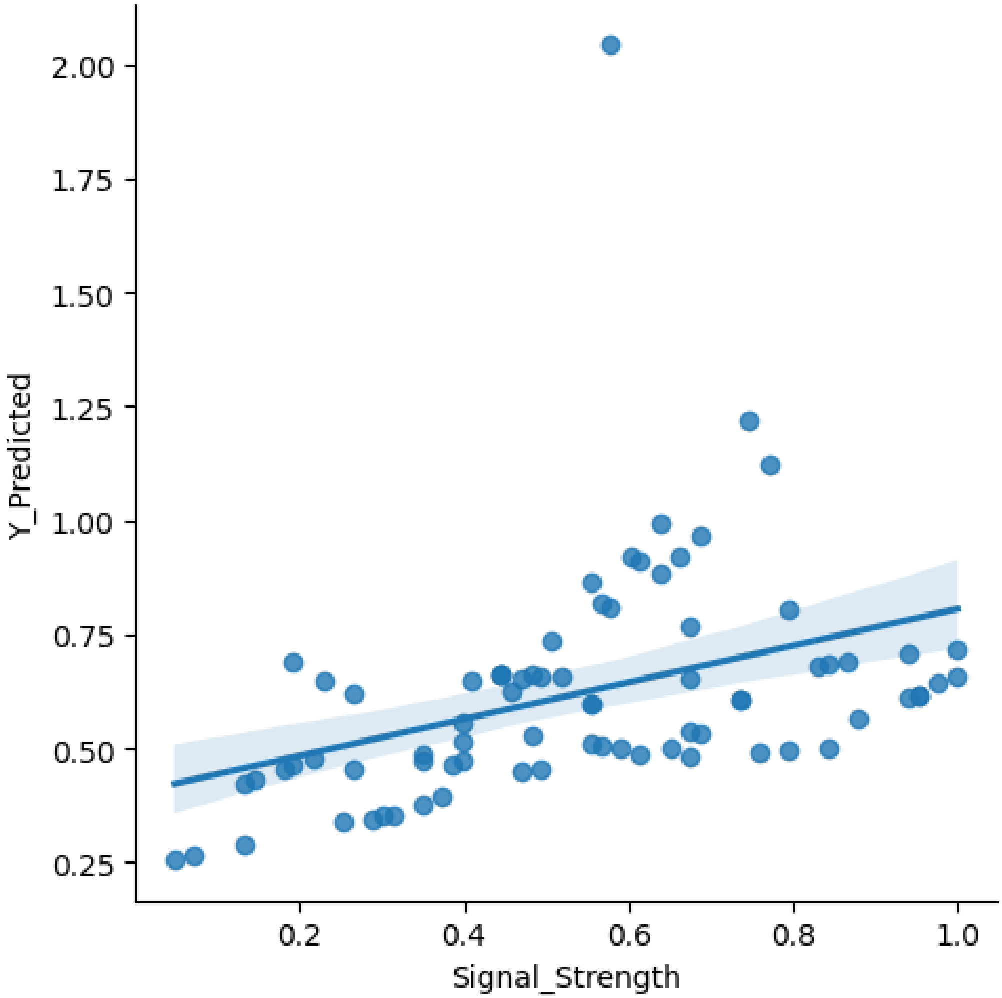
```
plt.scatter(y_test, lin_regressor_y_pred)
plt.show()
```

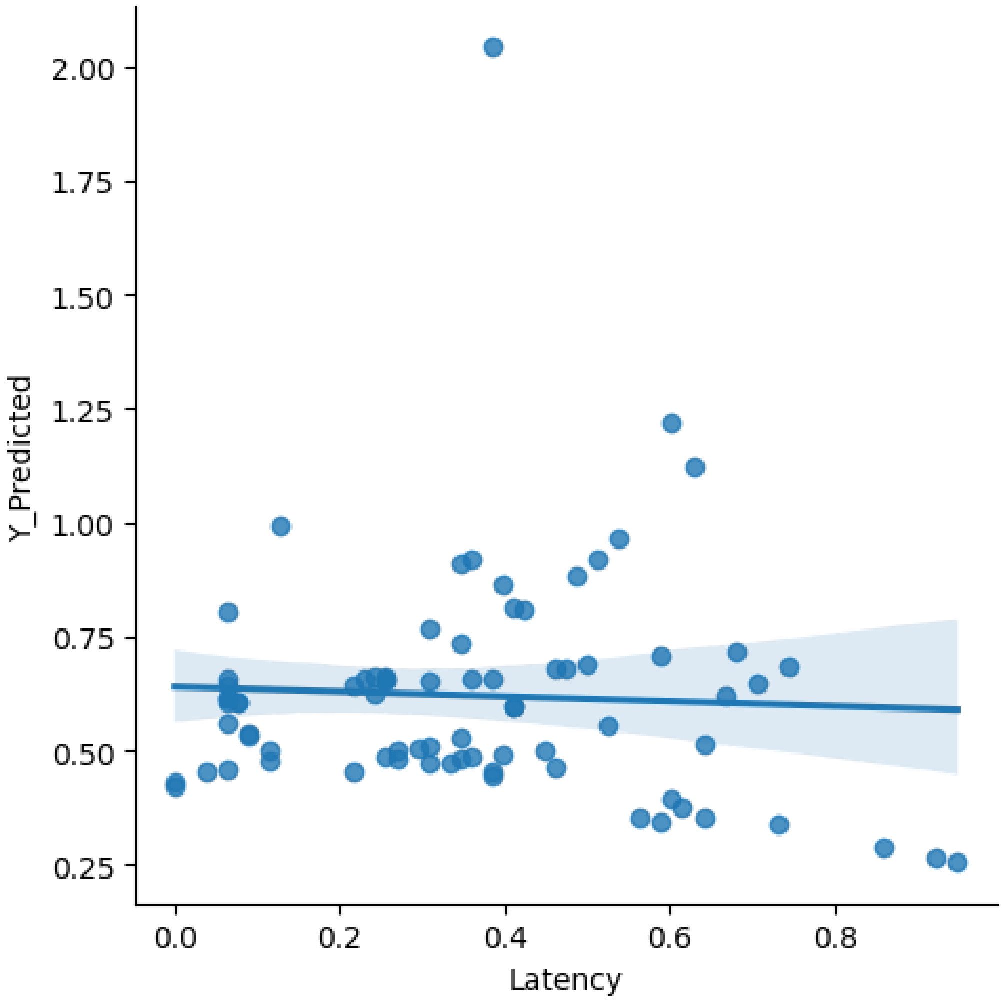


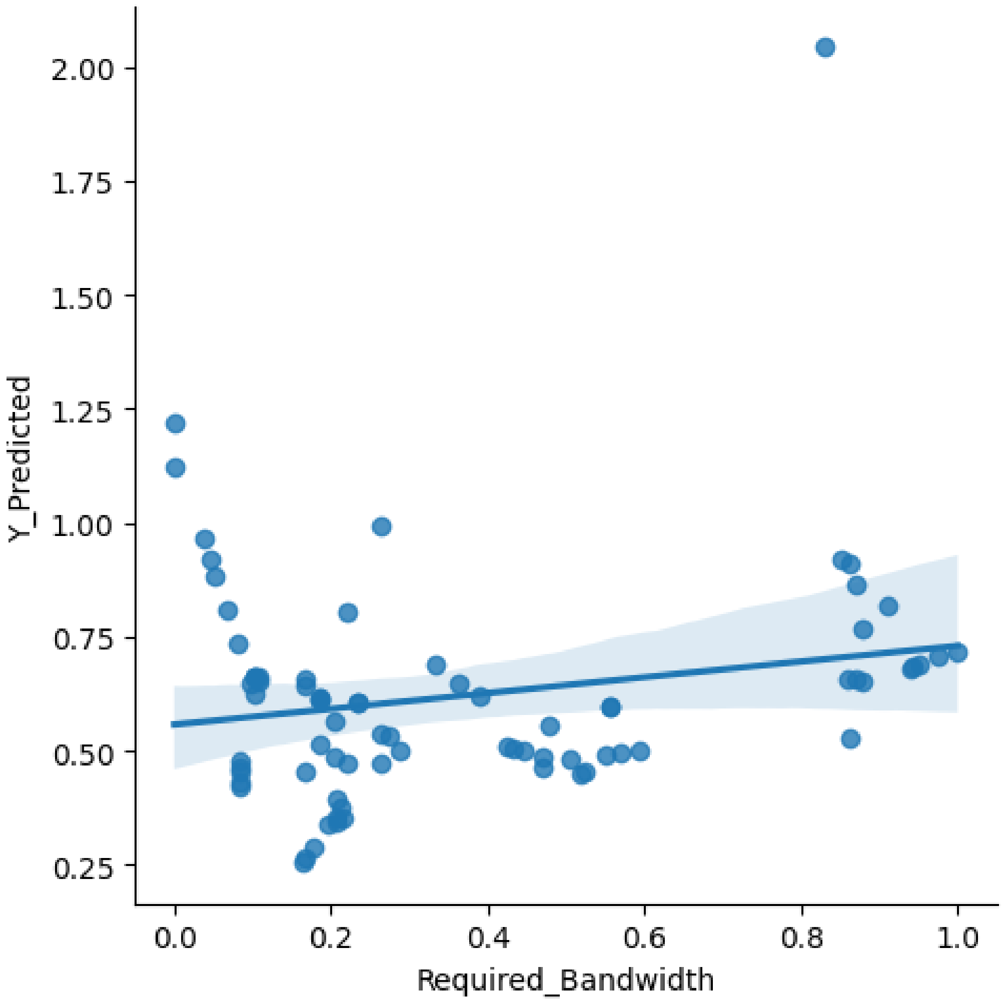
In [237]:

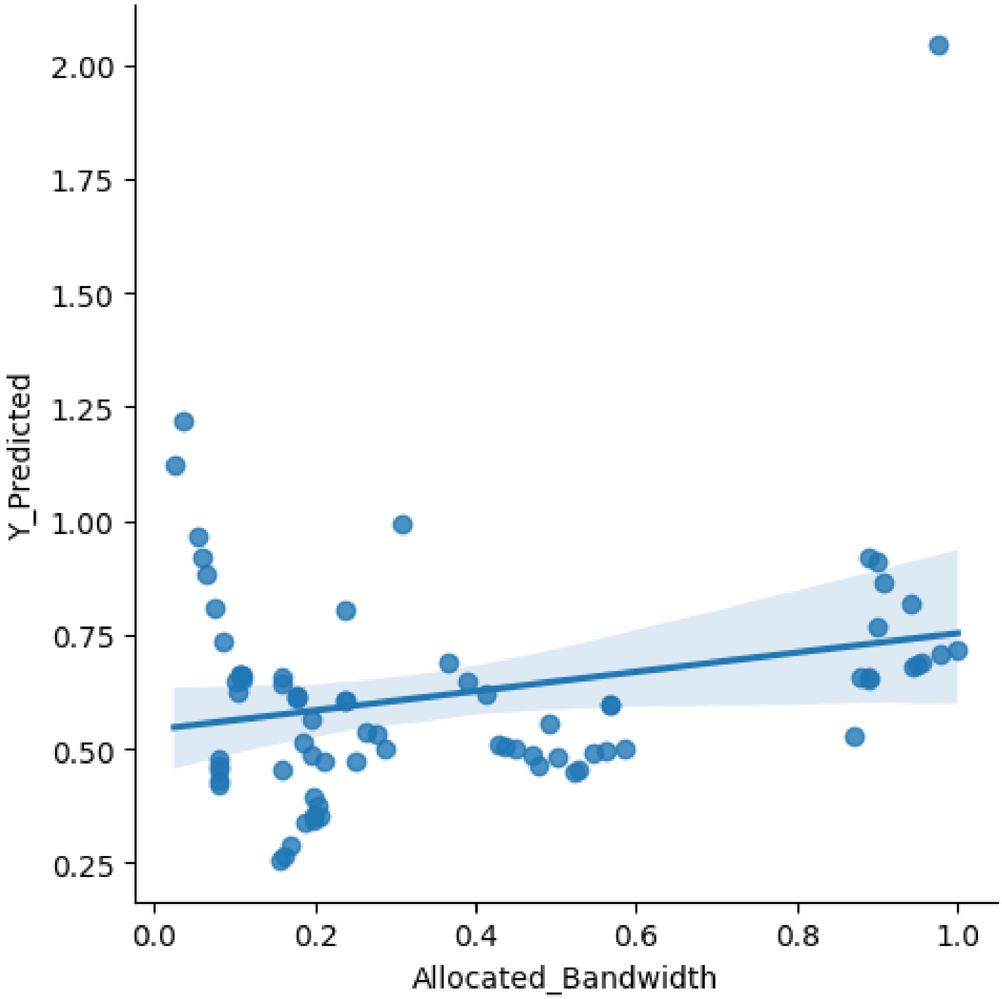
```
# Scatter Plots of each feature vs predicted value with regression line

lin_regressor_plot = x_test
lin_regressor_plot['Y_Predicted'] = lin_regressor_y_pred

sns.lmplot(x='Signal_Strength', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Latency', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Required_Bandwidth', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Allocated_Bandwidth', y='Y_Predicted', data=lin_regressor_plot)

plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```




2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Y_Predicted
0.0
0.2
0.4
0.6
0.8
Latency


Y_Predicted
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.0
0.2
0.4
0.6
0.8
1.0
Required_Bandwidth



SVM Regression Model :

Create and Train the Model :

In [238]:

```
from sklearn.svm import SVR
```

In [239]:

```
svr = SVR(kernel='poly')
svr.fit(x_train, y_train)
```

Out[239]:

```
▾ SVR
SVR(kernel='poly')
```

Predict Test Set Results :

In [240]:

```
x_test.drop('Y_Predicted', axis=1, inplace=True)
```

In [241]:

```
svr_y_pred = svr.predict(x_test)
```

Evaluation of Model Performance :

In [242]:

```
svr_r2 = r2_score(y_test, svr_y_pred)
svr_r2
```

Out[242]:

```
0.5166550618405741
```

In [243]:

```
svr_mse = mean_squared_error(y_test, svr_y_pred)
svr_mse
```

Out[243]:

```
0.035262344389530756
```

In [244]:

```
svr_rmse = np.sqrt(svr_mse)
svr_rmse
```

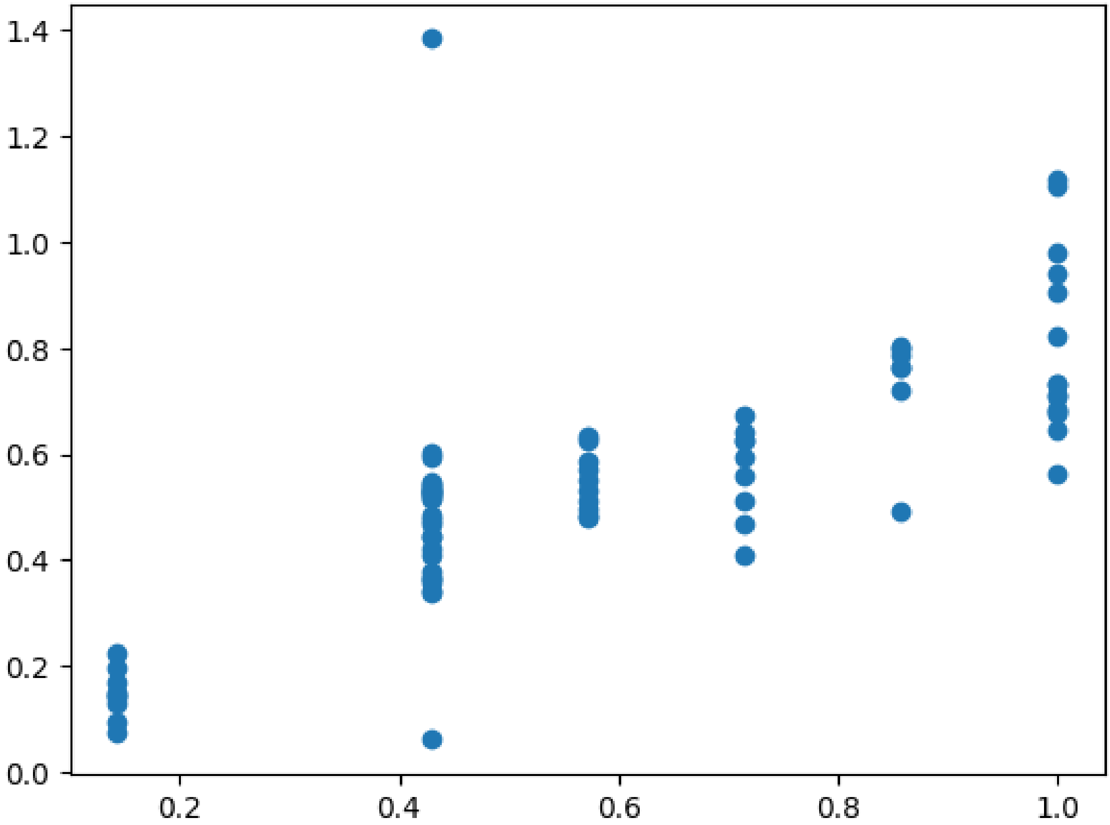
Out[244]:

```
0.1877827052460656
```

Visualize the Results of the Prediction :

In [245]:

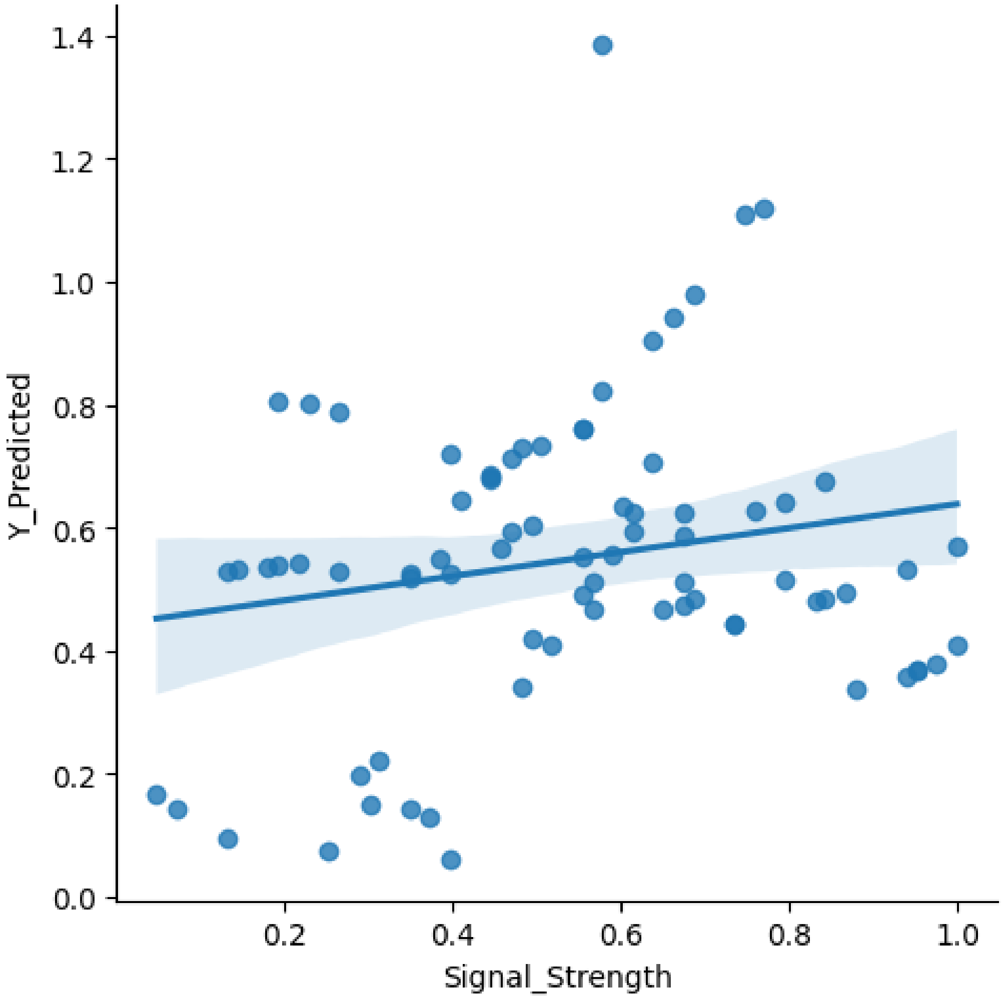
```
# Plot Actual vs Predicted Values

plt.scatter(y_test, svr_y_pred)
plt.show()
```

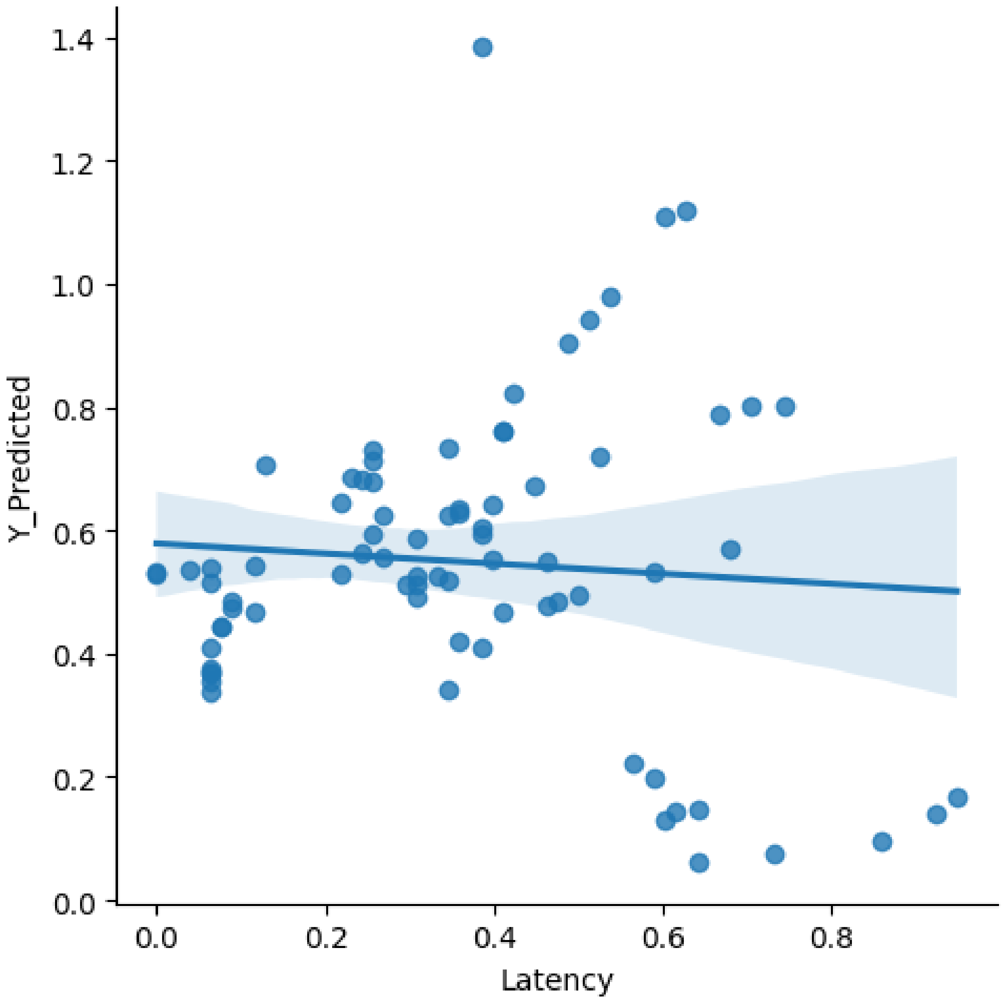


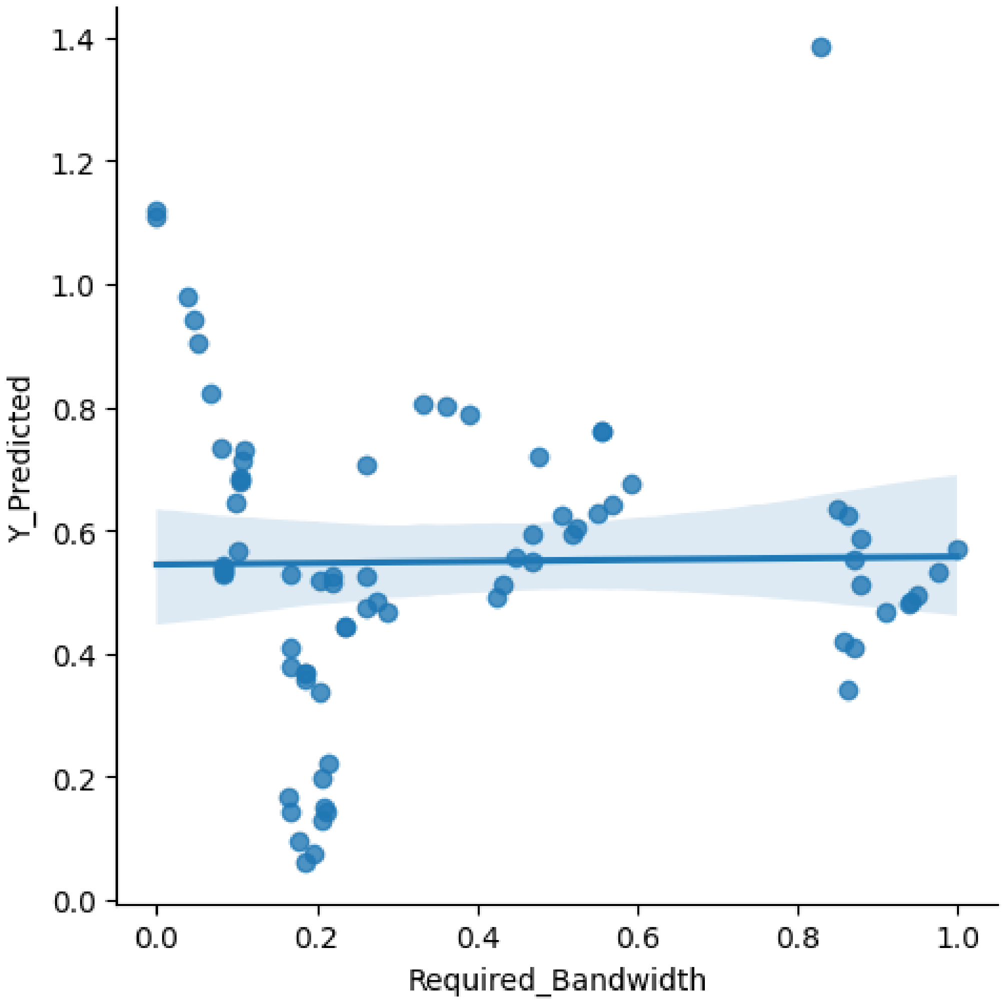
In [246]:

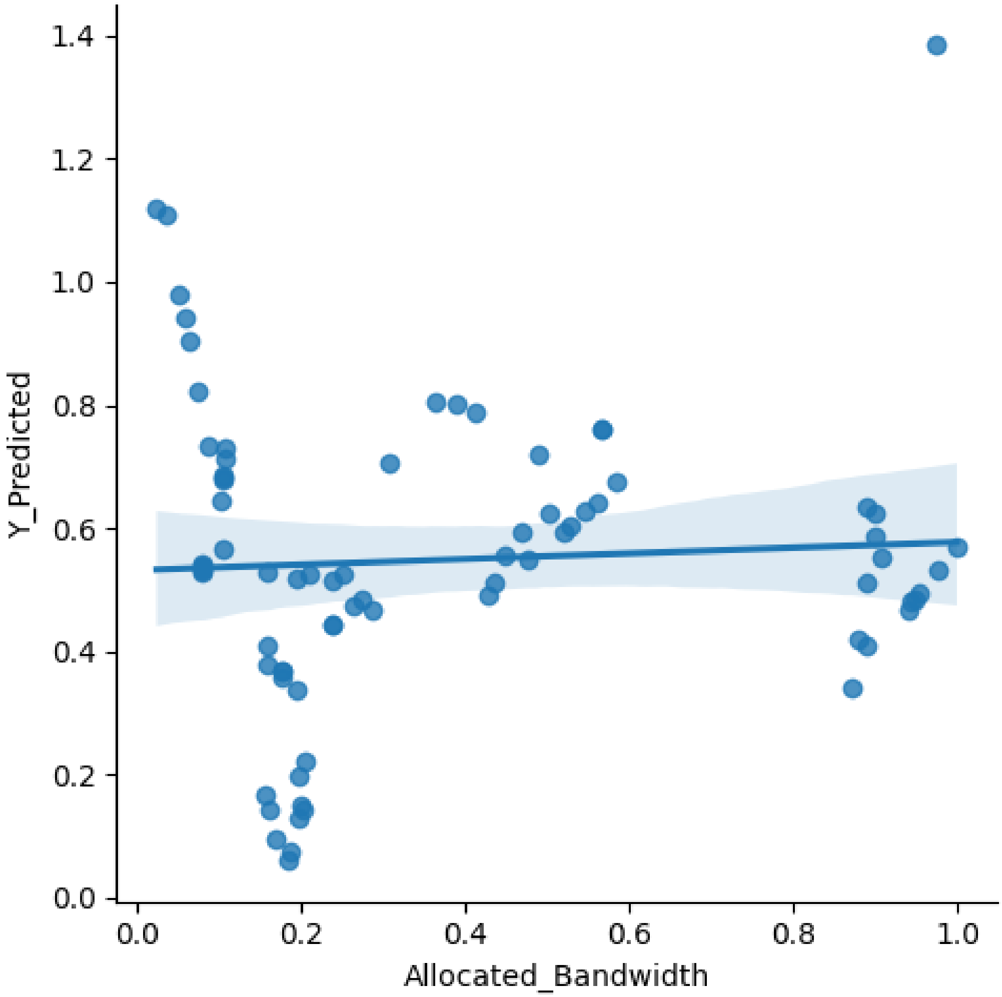
```
# Scatter Plots of each feature vs predicted value with regression line

lin_regressor_plot = x_test
lin_regressor_plot['Y_Predicted'] = svr_y_pred

sns.lmplot(x='Signal_Strength', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Latency', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Required_Bandwidth', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Allocated_Bandwidth', y='Y_Predicted', data=lin_regressor_plot)

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

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/tmp/ipykernel_32/4005064551.py:14: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```




1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Y_Predicted
0.0
0.2
0.4
0.6
0.8
Latency


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Y_Predicted
0.0
0.2
0.4
0.6
0.8
1.0
Required_Bandwidth



KNN Regression Model

Create and Train the Model :

In [247]:

```
from sklearn.neighbors import KNeighborsRegressor
```

In [248]:

```
knn_regr = KNeighborsRegressor(n_neighbors=5)
knn_regr.fit(x_train, y_train)
```

Out[248]:

```
▾ KNeighborsRegressor
KNeighborsRegressor()
```

Predict Test Set Results :

In [249]:

```
x_test.drop('Y_Predicted', axis=1, inplace=True)
```

In [250]:

```
knn_y_pred = svr.predict(x_test)
```

Evaluation of Model Performance :

In [251]:

```
knn_r2 = r2_score(y_test, knn_y_pred)
knn_r2
```

Out[251]:

```
0.5166550618405741
```

In [252]:

```
knn_mse = mean_squared_error(y_test, knn_y_pred)
knn_mse
```

Out[252]:

```
0.035262344389530756
```

In [253]:

```
knn_rmse = np.sqrt(knn_mse)
knn_rmse
```

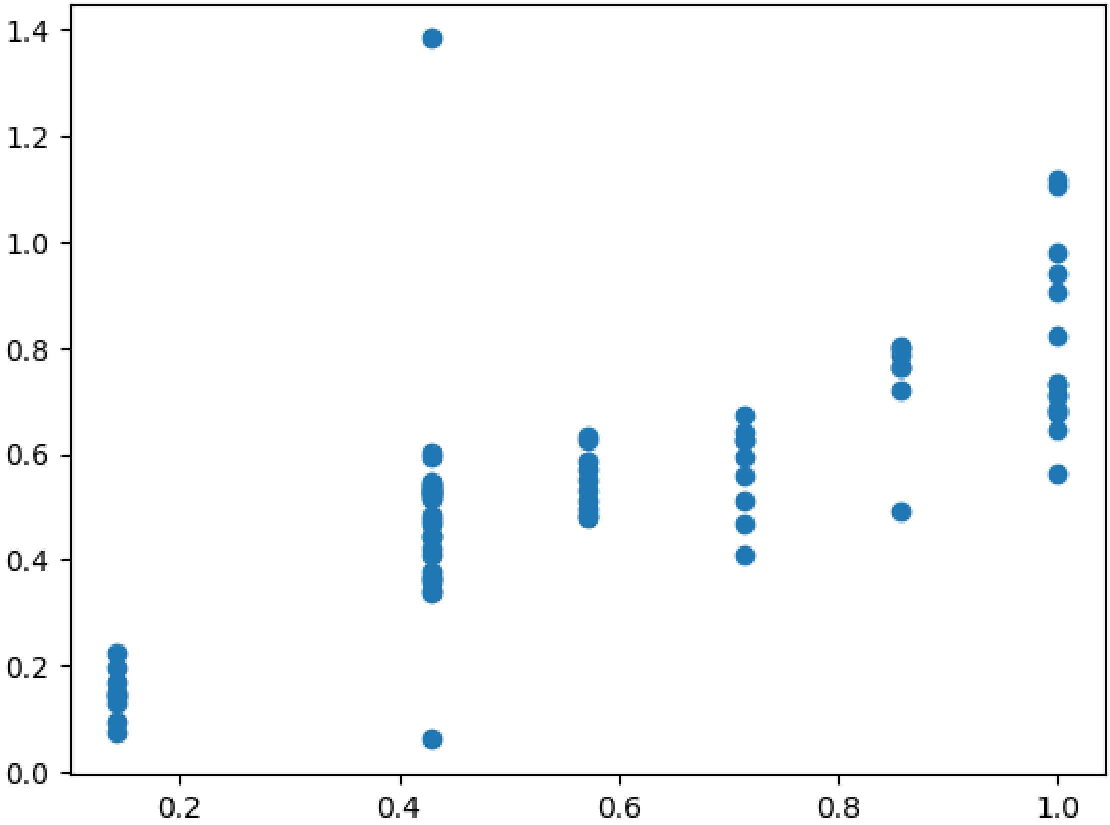
Out[253]:

```
0.1877827052460656
```

Visualize the Results of the Prediction :

In [254]:

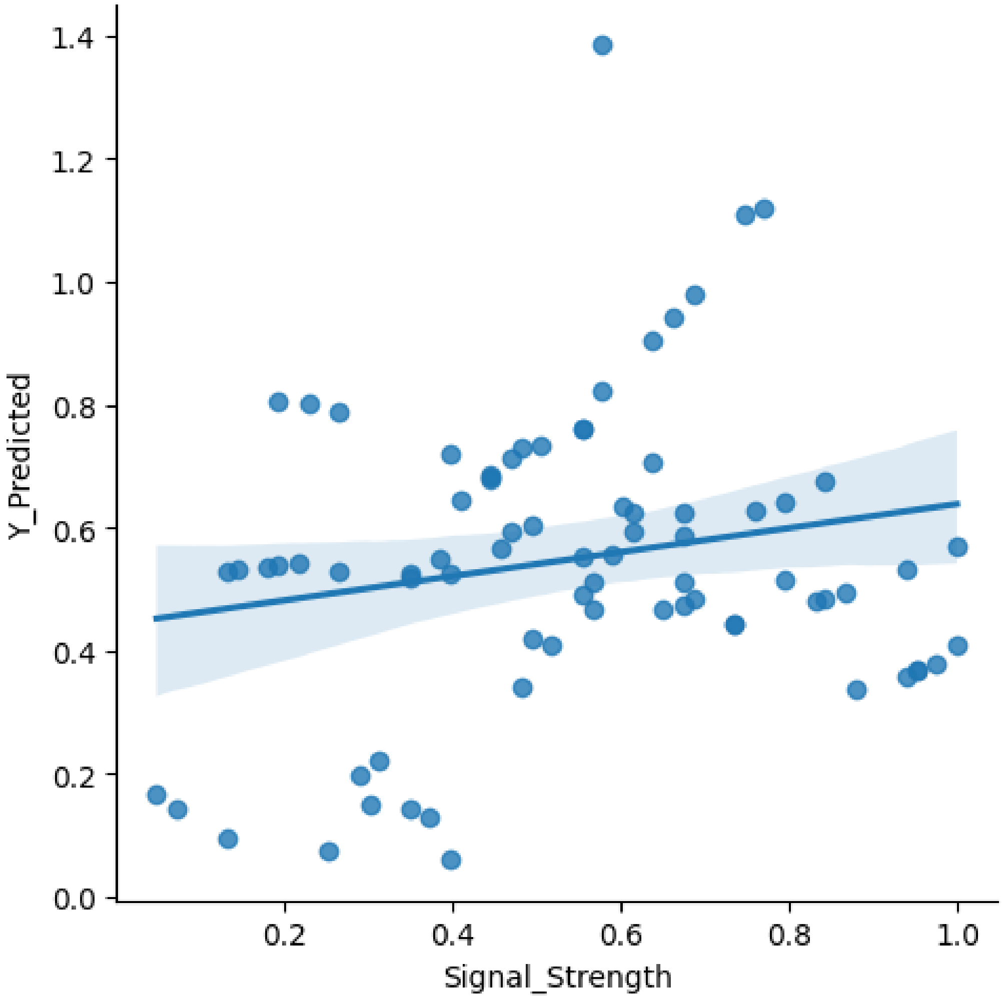
```
# Plot Actual vs Predicted Values

plt.scatter(y_test, knn_y_pred)
plt.show()
```

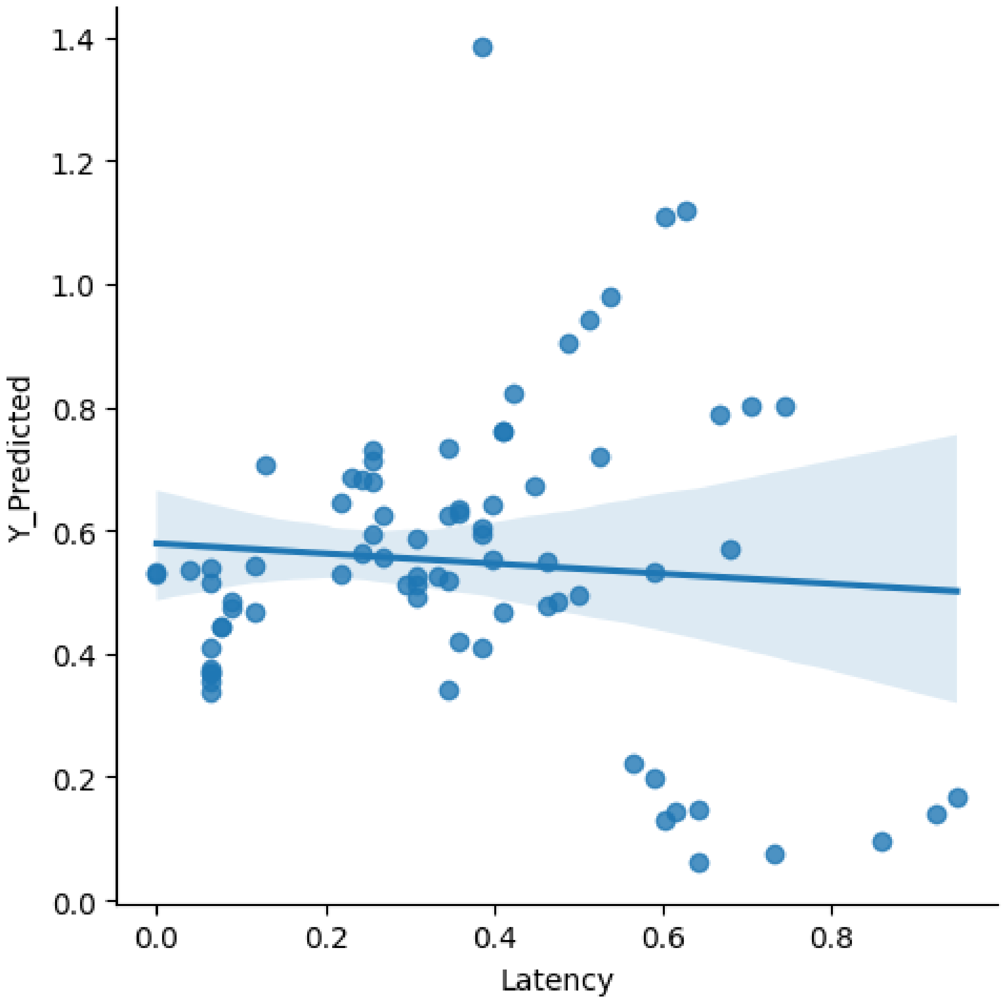


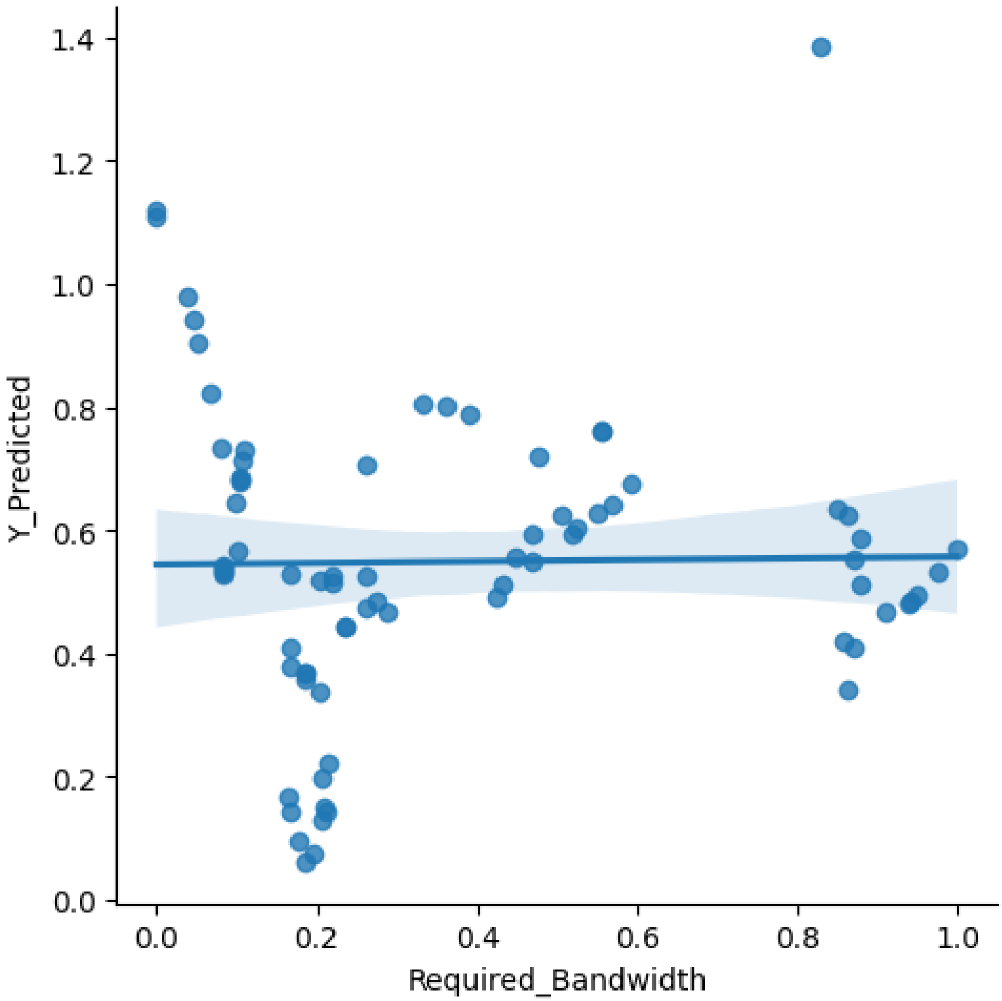
In [255]:

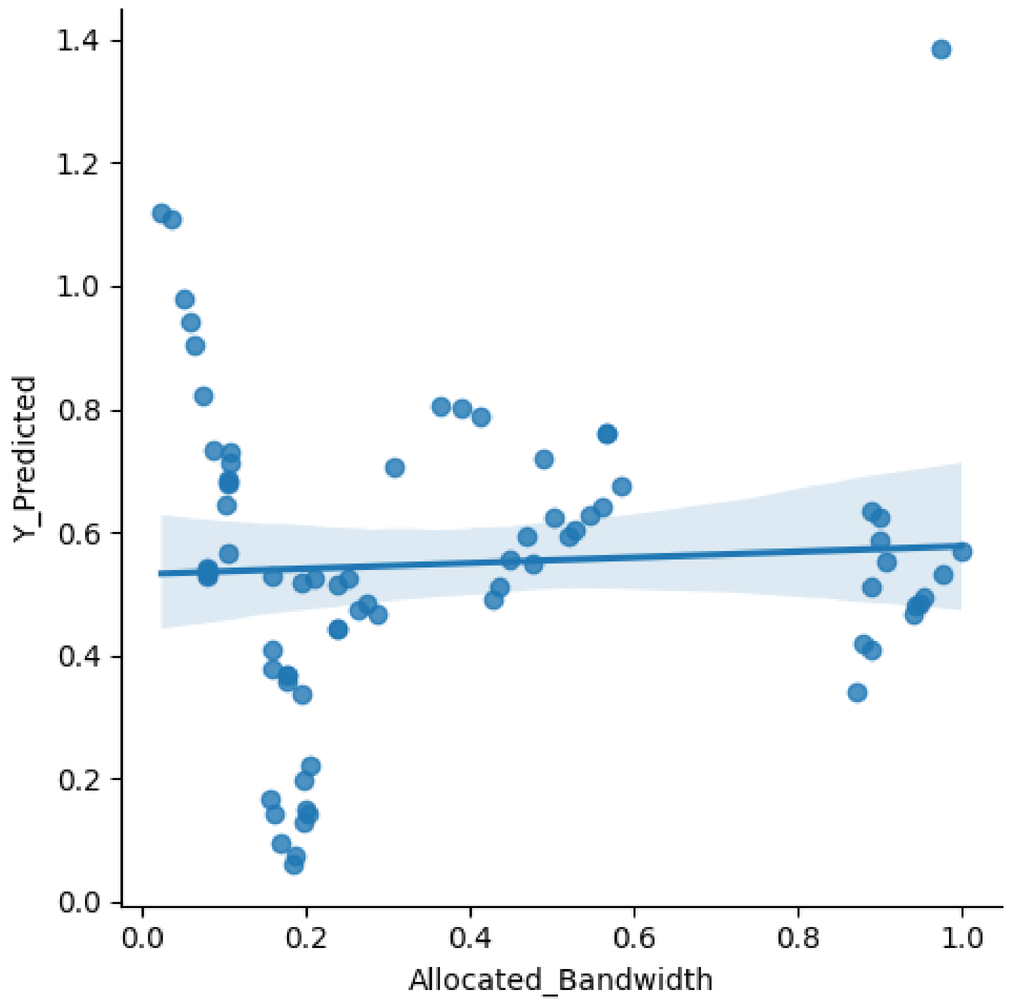
```
# Scatter Plots of each feature vs predicted value with regression line

lin_regressor_plot = x_test
lin_regressor_plot['Y_Predicted'] = knn_y_pred

sns.lmplot(x='Signal_Strength', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Latency', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Required_Bandwidth', y='Y_Predicted', data=lin_regressor_plot)

sns.lmplot(x='Allocated_Bandwidth', y='Y_Predicted', data=lin_regressor_plot)

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

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserW
arning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/tmp/ipykernel_32/1444102090.py:14: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```




1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Y_Predicted
0.0
0.2
0.4
0.6
0.8
Latency

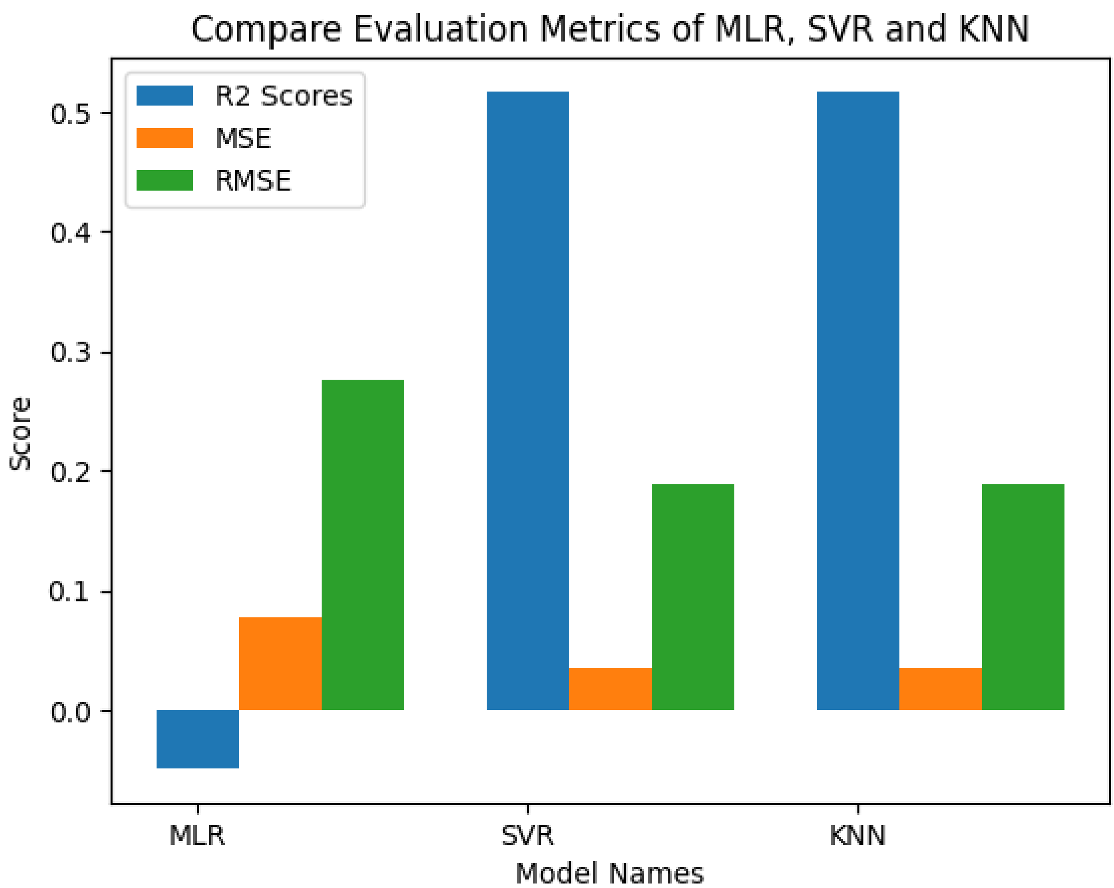

1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Y_Predicted
0.0
0.2
0.4
0.6
0.8
1.0
Required_Bandwidth



Comparing Results of MLR, SVM and KNN Regression Models

In [256]:

```
# Create the plot with multiple lines to compare measures like: mse, rmse and r2_sc
ore

categories = ['MLR', 'SVR', 'KNN']
x = np.arange(len(categories))
r2_scores = [lin_regressor_r2, svr_r2, knn_r2]
mse_scores = [lin_regressor_mse, svr_mse, knn_mse]
rmse_scores = [lin_regressor_rmse, svr_rmse, knn_rmse]

width = 0.25
fig, ax = plt.subplots()

ax.bar(x, r2_scores, label='R2 Scores', width=0.25)
ax.bar(x+width, mse_scores, label='MSE', width=0.25)
ax.bar(x+2*width, rmse_scores, label='RMSE', width=0.25)

# Customize the plot
plt.xlabel('Model Names')
plt.ylabel('Score')
plt.title('Compare Evaluation Metrics of MLR, SVR and KNN')
plt.legend()  # Show legend with labels
ax.set_xticks(x)
ax.set_xticklabels(categories)

# Show the plot
plt.show()
```



In [257]:

```
# Thus, we can conclude that, SVR and KNN Regressor has similar performance
# Which is better than MLR(Multliple Linear Regression)
```