mlp

December 6, 2025

[1]:
```
#Imports
import sys #Python
import sklearn #Machine learning library
import numpy as np #numerical packages in python
import scipy as scp #Another numerical package, unused directly but is␣
 ↪implicitly used in sklearn
import pandas as pd #Package for data manipulation and analysis
import matplotlib.pyplot as plt # plotting library
import os
import time
import random
import math

# SKlearn imports
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
```

[2]:
```
#Load Data
data = pd.read_csv('./pathloss_data.csv')
data.head()
```

[2]:
```
   tx_lat     tx_lon       tx_el   rx_lat     rx_lon       rx_el  \
0   34.07 -118.44270  112.617782  34.0674 -118.44225  109.278807
1   34.07 -118.44265  112.807114  34.0674 -118.44225  109.278807
2   34.07 -118.44260  112.996442  34.0674 -118.44225  109.278807
3   34.07 -118.44255  113.185765  34.0674 -118.44225  109.278807
4   34.07 -118.44250  113.375084  34.0674 -118.44225  109.278807

   frequency_Hz  pathloss_dB  pl_los_index  propagation_delay_s  \
0    2400000000   137.434941             1             0.000001
1    2400000000   137.035171             1             0.000001
2    2400000000   136.609911             1             0.000001
3    2400000000   136.155890             1             0.000001
```

```
4    2400000000   135.669330            1            0.000001

   propagation_distance_m  rays_count  h_bar  SNR_dB  capacity     loss_dB
0              343.431802          44    0.0    -inf       0.0  137.434947
1              339.071364          45    0.0    -inf       0.0  137.035177
2              334.744002          47    0.0    -inf       0.0  136.609917
3              330.456330          59    0.0    -inf       0.0  136.155896
4              326.216763          76    0.0    -inf       0.0  135.669336
```

[3]:
```
# Remove entries that have -inf or inf values
data = data.replace([np.inf, -np.inf], np.nan).dropna()

# Replace entries with a pl_los_index greater than 1 with 0
data.loc[data["pl_los_index"] > 1, "pl_los_index"] = 0
```

[4]:
```
data.head()
```

[4]:
```
    tx_lat     tx_lon       tx_el   rx_lat     rx_lon       rx_el  \
9    34.07 -118.44225  114.321714  34.0674 -118.44225  109.278807
10   34.07 -118.44220  114.495168  34.0674 -118.44225  109.278807
11   34.07 -118.44215  114.571634  34.0674 -118.44225  109.278807
12   34.07 -118.44210  114.648095  34.0674 -118.44225  109.278807
13   34.07 -118.44205  114.724552  34.0674 -118.44225  109.278807

    frequency_Hz  pathloss_dB  pl_los_index  propagation_delay_s  \
9     2400000000   120.808919             0         9.860568e-07
10    2400000000   110.300298             0         9.813725e-07
11    2400000000   103.518610             0         1.040409e-06
12    2400000000    93.178707             0         9.750255e-07
13    2400000000    89.271412             1         9.641608e-07

    propagation_distance_m  rays_count     h_bar     SNR_dB      capacity  \
9               295.612405          37  0.018786 -32.485389  2.872640e-07
10              294.208084          38  0.080248 -10.780841  7.759651e-04
11              311.906721          37  0.115459  -1.654201  1.308086e-02
12              292.305294          21  0.099946   6.907486  6.902770e-02
13              289.048151          22  0.111347  10.261359  1.784548e-01

       loss_dB
9   120.808925
10  110.300303
11  103.518615
12   93.178712
13   89.271418
```

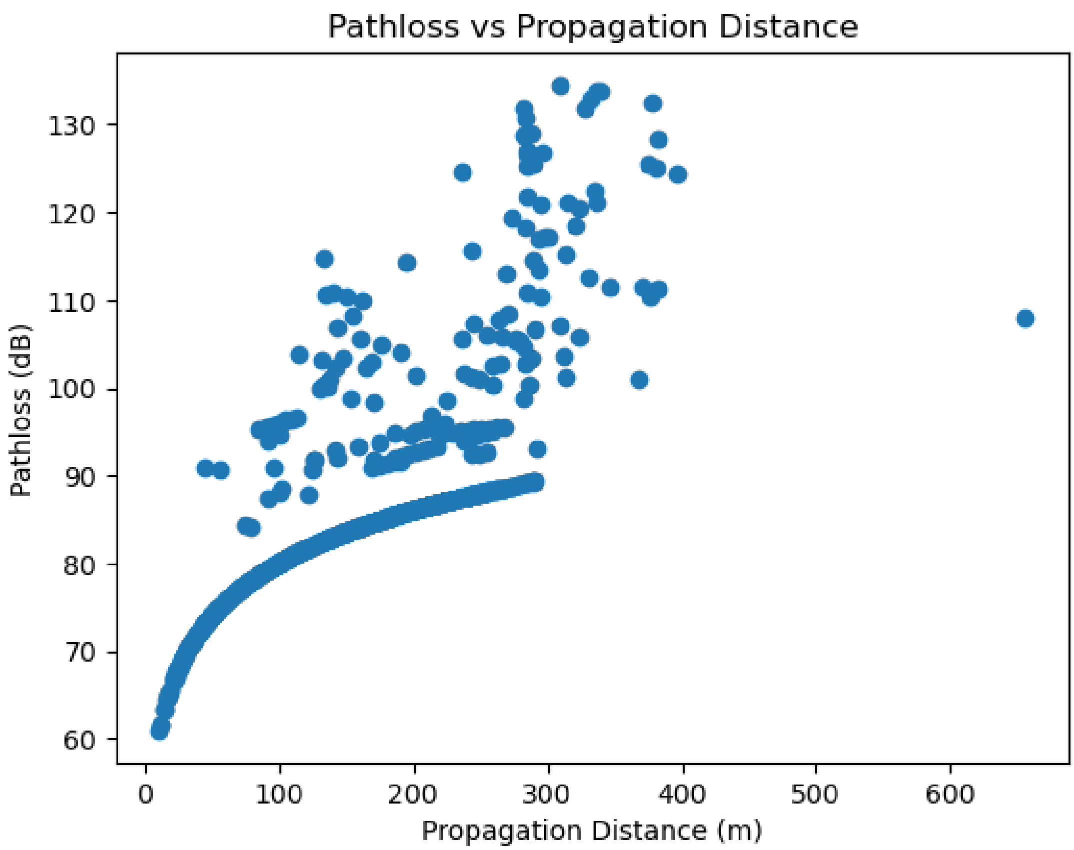
[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Index: 969 entries, 9 to 1019
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   tx_lat                  969 non-null    float64
 1   tx_lon                  969 non-null    float64
 2   tx_el                   969 non-null    float64
 3   rx_lat                  969 non-null    float64
 4   rx_lon                  969 non-null    float64
 5   rx_el                   969 non-null    float64
 6   frequency_Hz            969 non-null    int64
 7   pathloss_dB             969 non-null    float64
 8   pl_los_index            969 non-null    int64
 9   propagation_delay_s     969 non-null    float64
 10  propagation_distance_m  969 non-null    float64
 11  rays_count              969 non-null    int64
 12  h_bar                   969 non-null    float64
 13  SNR_dB                  969 non-null    float64
 14  capacity                969 non-null    float64
 15  loss_dB                 969 non-null    float64
dtypes: float64(13), int64(3)
memory usage: 128.7 KB
```

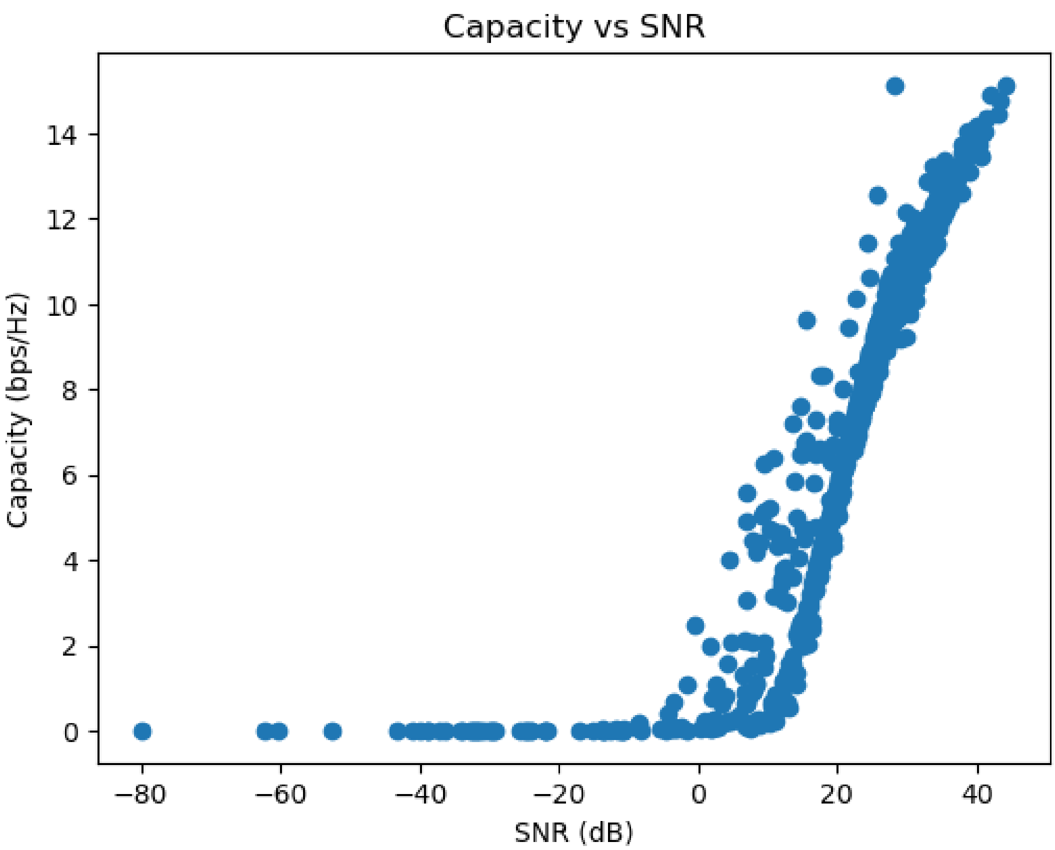
[6]:

```
plt.plot(data['propagation_distance_m'], data['pathloss_dB'], 'o')
plt.xlabel('Propagation Distance (m)')
plt.ylabel('Pathloss (dB)')
plt.title('Pathloss vs Propagation Distance')
plt.show()
```



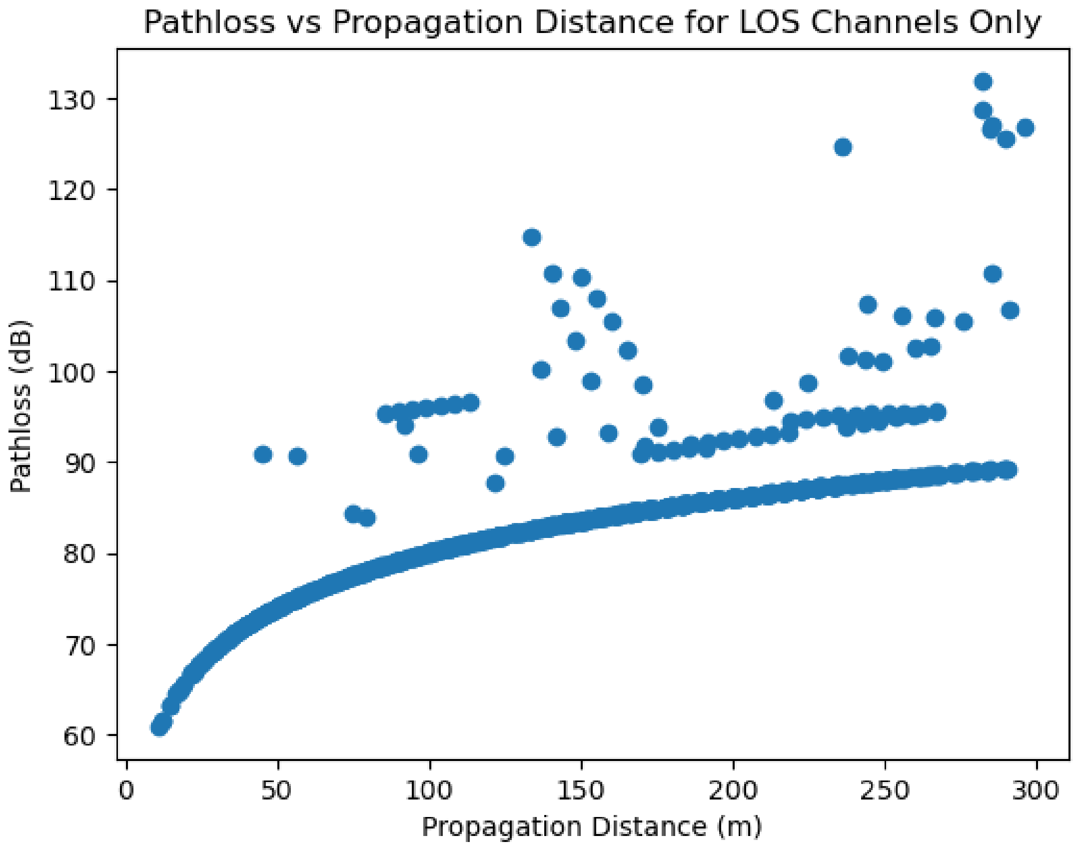
[7]:

```
plt.plot(data['SNR_dB'], data['capacity'], 'o')
plt.xlabel('SNR (dB)')
plt.ylabel('Capacity (bps/Hz)')
plt.title('Capacity vs SNR')
plt.show()
```

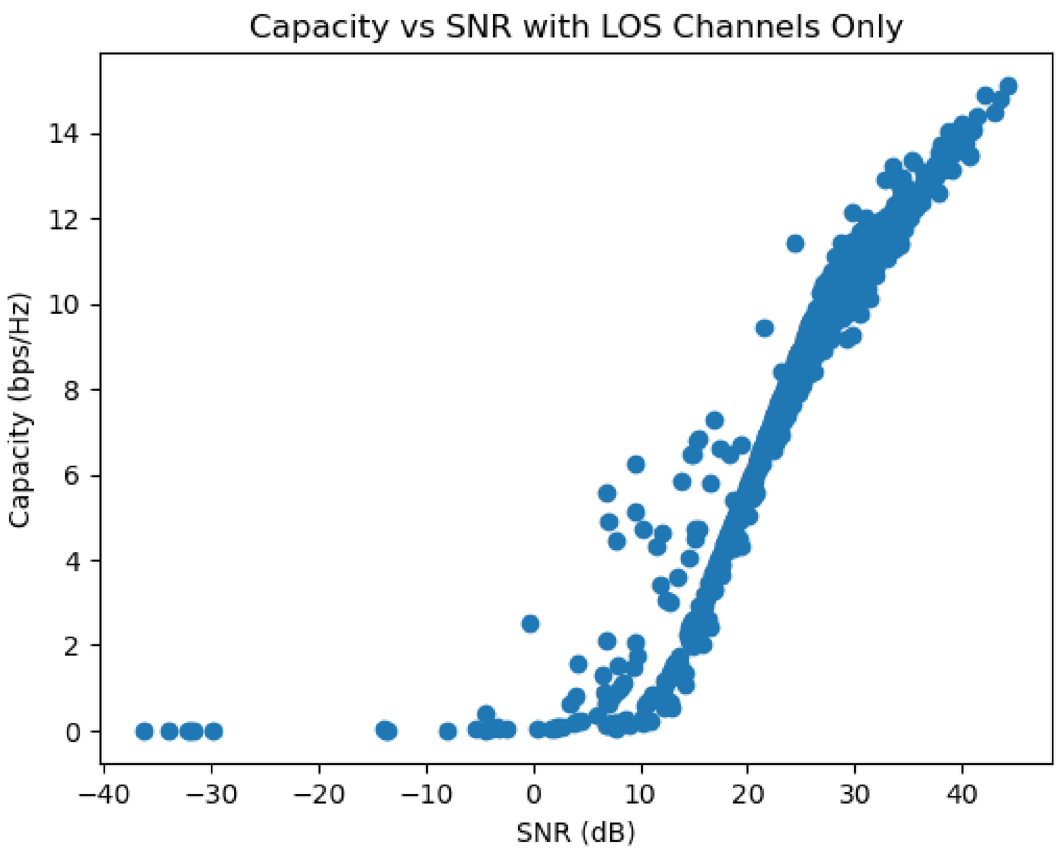


[8]:
```
dataLOS = data[data['pl_los_index'] == 1]
```

[9]:
```
plt.plot(dataLOS['propagation_distance_m'], dataLOS['pathloss_dB'], 'o')
plt.xlabel('Propagation Distance (m)')
plt.ylabel('Pathloss (dB)')
plt.title('Pathloss vs Propagation Distance for LOS Channels Only')
plt.show()
```



[10]:
```
plt.plot(dataLOS['SNR_dB'], dataLOS['capacity'], 'o')
plt.xlabel('SNR (dB)')
plt.ylabel('Capacity (bps/Hz)')
plt.title('Capacity vs SNR with LOS Channels Only')
plt.show()
```



[11]:
```python
#Preprocess Data
data = data.drop("frequency_Hz", axis=1)
data = data.drop("rays_count", axis=1)
data = data.drop('loss_dB', axis=1)
data = data.drop('SNR_dB', axis=1)
data = data.drop('h_bar', axis=1)
data = data.drop('propagation_delay_s', axis=1)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 969 entries, 9 to 1019
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   tx_lat                  969 non-null    float64
 1   tx_lon                  969 non-null    float64
 2   tx_el                   969 non-null    float64
 3   rx_lat                  969 non-null    float64
 4   rx_lon                  969 non-null    float64
 5   rx_el                   969 non-null    float64
```

```
 6   pathloss_dB             969 non-null    float64
 7   pl_los_index            969 non-null    int64
 8   propagation_distance_m  969 non-null    float64
 9   capacity                969 non-null    float64
dtypes: float64(9), int64(1)
memory usage: 83.3 KB
```

[12]:

```
R = 6371000  # Earth radius in meters

def north_south_distance(lat1, lat2):
    """
    Compute north-south distance (meters) due only to change in latitude.
    Positive if lat2 is north of lat1.
    """
    dlat = math.radians(lat2 - lat1)
    return dlat * R


def east_west_distance(lat1, lon1, lon2):
    """
    Compute east-west distance (meters) due only to change in longitude.
    Positive if lon2 is east of lon1.
    Uses the latitude (lat1) to scale longitude distance.
    """
    dlon = math.radians(lon2 - lon1)
    lat_rad = math.radians(lat1)
    return dlon * R * math.cos(lat_rad)

def elevation_difference(ele1, ele2):
    """
    Compute elevation difference (meters).
    Positive if ele2 is above ele1.
    """
    return ele2 - ele1

data['ns_distance'] = data.apply(lambda row:
 ↪north_south_distance(row['rx_lat'], row['tx_lat']), axis=1)
data['ew_distance'] = data.apply(lambda row: east_west_distance(row['tx_lat'],
 ↪row['rx_lon'], row['tx_lon']), axis=1)
data['elevation_diff'] = data.apply(lambda row:
 ↪elevation_difference(row['rx_el'], row['tx_el']), axis=1)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 969 entries, 9 to 1019
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
```

```
 0   tx_lat                  969 non-null    float64
 1   tx_lon                  969 non-null    float64
 2   tx_el                   969 non-null    float64
 3   rx_lat                  969 non-null    float64
 4   rx_lon                  969 non-null    float64
 5   rx_el                   969 non-null    float64
 6   pathloss_dB             969 non-null    float64
 7   pl_los_index            969 non-null    int64
 8   propagation_distance_m  969 non-null    float64
 9   capacity                969 non-null    float64
 10  ns_distance             969 non-null    float64
 11  ew_distance             969 non-null    float64
 12  elevation_diff          969 non-null    float64
dtypes: float64(12), int64(1)
memory usage: 106.0 KB
```

[13]:
```
data = data.drop(['tx_lat', 'tx_lon', 'tx_el', 'rx_lat', 'rx_lon', 'rx_el'],␣
 ↪axis=1)
data = data.drop('propagation_distance_m', axis=1)
data.head(20)
```

[13]:
```
    pathloss_dB  pl_los_index      capacity  ns_distance  ew_distance  \
9    120.808919             0  2.872640e-07   289.106809     0.000000
10   110.300298             0  7.759651e-04   289.106809     4.605437
11   103.518610             0  1.308086e-02   289.106809     9.210874
12    93.178707             0  6.902770e-02   289.106809    13.816311
13    89.271412             1  1.784548e-01   289.106809    18.421747
14    89.281411             1  5.179047e-01   289.106809    23.027184
15    89.293591             1  2.357805e-01   289.106809    27.632621
16    89.307935             1  5.135356e-02   289.106809    32.238058
17   106.667927             1  1.401035e-04   289.106809    36.843495
18   111.467517             0  3.022812e-06   289.106809    41.448932
19   112.606535             0  9.166257e-08   289.106809    46.054368
28   126.748164             1  2.916656e-07   283.547063    -4.605440
29   122.508611             0  2.015742e-06   283.547063     0.000000
30   114.545027             0  1.104685e-03   283.547063     4.605440
31   103.374928             0  7.337151e-02   283.547063     9.210879
32   100.442864             0  2.256317e-01   283.547063    13.816319
33    89.103450             1  6.717699e-01   283.547063    18.421758
34    89.113841             1  1.088342e+00   283.547063    23.027198
35    89.126499             1  5.512226e-01   283.547063    27.632637
36    89.141403             1  1.371613e-01   283.547063    32.238077

    elevation_diff
9         5.042907
10        5.216362
11        5.292827
```

```
12         5.369288
13         5.445745
14         5.522198
15         5.598654
16         5.675121
17         5.751583
18         5.828034
19         5.904496
28         4.860864
29         5.034315
30         5.110785
31         5.187235
32         5.263704
33         5.340161
34         5.416606
35         5.493069
36         5.569528
```

[14]:
```
#Split target and data
data_pathloss = data["pathloss_dB"]
data_capacity = data["capacity"]
data_geo = data.drop(["pathloss_dB", "capacity"], axis = 1)
```

[15]:
```
# Train an MLP for pathloss
#Train test split
train_raw, test_raw, target_pl, target_pl_test = train_test_split(data_geo,␣
 ↪data_pathloss, test_size=0.2, random_state=0)
```

[16]:
```
#Standardize data
#Since all features are real-valued, we only have one pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler())
])

#Transform raw data
train = pipeline.fit_transform(train_raw)
test = pipeline.transform(test_raw) #Note that there is no fit calls

#Names of Features after Pipeline
feature_names = list(pipeline.get_feature_names_out(list(data_geo.columns)))
```

[17]:
```
regr_pl = MLPRegressor(hidden_layer_sizes=(100,), max_iter = 100000)
regr_pl.fit(train, target_pl)
predicted_pl = regr_pl.predict(test)
```

[18]:
```
print("%-12s %f" % ('Accuracy:', r2_score(target_pl_test,predicted_pl)))
```

```
Accuracy:    0.858586
```

[19]:
```
# Now train an MLP for capacity
train_raw, test_raw, target_cap, target_cap_test = train_test_split(data_geo,␣
↪data_capacity, test_size=0.2, random_state=0)
```

[20]:
```
#Standardize data
#Since all features are real-valued, we only have one pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler())
])

#Transform raw data
train = pipeline.fit_transform(train_raw)
test = pipeline.transform(test_raw) #Note that there is no fit calls

#Names of Features after Pipeline
feature_names = list(pipeline.get_feature_names_out(list(data_geo.columns)))
```

[21]:
```
regr_cap = MLPRegressor(hidden_layer_sizes=(100,), max_iter = 100000)
regr_cap.fit(train, target_cap)
predicted_cap = regr_cap.predict(test)
```

[22]:
```
print("%-12s %f" % ('Accuracy:', r2_score(target_cap_test,predicted_cap)))
```

```
Accuracy:    0.868022
```

[23]:
```
target_cap_test
```

[23]:
```
983    11.424888
330     7.303439
31      0.073372
732     9.963909
509     6.397362
          …
631     9.065862
540     5.410754
35      0.551223
471     7.100764
550     6.798059
Name: capacity, Length: 194, dtype: float64
```

[24]:
```
# MCS Table
rlog2M = [0, 0.194, 0.248, 0.312, 0.401, 0.500, 0.618, 0.737, 0.856, 0.957, 1.
↪075, 1.233, 1.411, 1.589, 1.767, 2.000, 2.089, 2.267, 2.533, 2.711, 2.944, 3.
↪181, 3.537, 3.780, 3.928, 4.225, 4.373, 5.070]
r = [0, 0.097, 0.124, 0.156, 0.206, 0.250, 0.309, 0.368, 0.428, 0.478, 0.538, 0.
↪617, 0.353, 0.397, 0.442, 0.500, 0.522, 0.567, 0.633, 0.678, 0.736, 0.795, 0.
↪589, 0.630, 0.655, 0.704, 0.729, 0.845]
```

```
M = [0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 16, 16, 16, 16, 16, 16, 16, 16, 16,␣
↪16, 16, 64, 64, 64, 64, 64, 64]
```

[25]:
```python
for i in range(len(predicted_cap)):
    mcs_idx = 0
    for j in range(len(rlog2M)):
        if predicted_cap[i] >= rlog2M[j]:
            mcs_idx = j

    print(f"Predicted Capacity: {predicted_cap[i]:.2f}, MCS Index: {mcs_idx}, M:
↪ {M[mcs_idx]}, r: {r[mcs_idx]:.3f}")
```

```
Predicted Capacity: 12.00, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.52, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 0.27, MCS Index: 2, M: 4, r: 0.124
Predicted Capacity: 9.95, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.23, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.70, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.87, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.59, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.19, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.62, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.80, MCS Index: 14, M: 16, r: 0.442
Predicted Capacity: 12.56, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.49, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.08, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.31, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.94, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.90, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.77, MCS Index: 14, M: 16, r: 0.442
Predicted Capacity: 7.15, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.02, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 2.78, MCS Index: 19, M: 16, r: 0.678
Predicted Capacity: 8.90, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.11, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.04, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.80, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.93, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 3.94, MCS Index: 24, M: 64, r: 0.655
Predicted Capacity: 8.87, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.10, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.91, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.34, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.16, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 2.89, MCS Index: 19, M: 16, r: 0.678
Predicted Capacity: 9.81, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 4.13, MCS Index: 24, M: 64, r: 0.655
Predicted Capacity: 11.81, MCS Index: 27, M: 64, r: 0.845
```

```
Predicted Capacity: 8.57, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.27, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.44, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.11, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 3.53, MCS Index: 21, M: 16, r: 0.795
Predicted Capacity: 3.13, MCS Index: 20, M: 16, r: 0.736
Predicted Capacity: 7.41, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.05, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.82, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.11, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.97, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 12.66, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.91, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.70, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: -0.81, MCS Index: 0, M: 0, r: 0.000
Predicted Capacity: 2.20, MCS Index: 16, M: 16, r: 0.522
Predicted Capacity: 9.95, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.64, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.74, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.72, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.93, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.67, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 12.07, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.27, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.05, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.80, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 3.28, MCS Index: 21, M: 16, r: 0.795
Predicted Capacity: 7.64, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.42, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 4.69, MCS Index: 26, M: 64, r: 0.729
Predicted Capacity: 7.15, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.97, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.32, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.70, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 0.49, MCS Index: 4, M: 4, r: 0.206
Predicted Capacity: 7.51, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.12, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.41, MCS Index: 12, M: 16, r: 0.353
Predicted Capacity: 10.74, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.00, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.40, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.86, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 6.56, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.34, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.60, MCS Index: 13, M: 16, r: 0.397
Predicted Capacity: 2.15, MCS Index: 16, M: 16, r: 0.522
Predicted Capacity: 12.72, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.41, MCS Index: 27, M: 64, r: 0.845
```

```
Predicted Capacity: 6.99, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 13.58, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 0.59, MCS Index: 5, M: 4, r: 0.250
Predicted Capacity: 12.36, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 6.98, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.18, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.74, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.75, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.34, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.81, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.64, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.40, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.29, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.28, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 12.37, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.78, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.84, MCS Index: 14, M: 16, r: 0.442
Predicted Capacity: 10.27, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.26, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.14, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: -0.36, MCS Index: 0, M: 0, r: 0.000
Predicted Capacity: 6.42, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.87, MCS Index: 14, M: 16, r: 0.442
Predicted Capacity: 7.22, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.49, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.37, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 13.17, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.24, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.25, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.68, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.66, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.79, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 13.50, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: -0.24, MCS Index: 0, M: 0, r: 0.000
Predicted Capacity: 8.19, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 2.30, MCS Index: 17, M: 16, r: 0.567
Predicted Capacity: 4.64, MCS Index: 26, M: 64, r: 0.729
Predicted Capacity: 8.29, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 4.82, MCS Index: 26, M: 64, r: 0.729
Predicted Capacity: 8.79, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.79, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.96, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 3.78, MCS Index: 22, M: 16, r: 0.589
Predicted Capacity: 0.45, MCS Index: 4, M: 4, r: 0.206
Predicted Capacity: 6.97, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.82, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 4.59, MCS Index: 26, M: 64, r: 0.729
Predicted Capacity: 7.76, MCS Index: 27, M: 64, r: 0.845
```

```
Predicted Capacity: 10.28, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.96, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.88, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.75, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 11.64, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.33, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.85, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.15, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 6.19, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.24, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.83, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 10.30, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 6.76, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.17, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.65, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 4.77, MCS Index: 26, M: 64, r: 0.729
Predicted Capacity: 10.66, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.97, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.38, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 12.01, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.36, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.84, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.50, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 9.60, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 12.31, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 3.64, MCS Index: 22, M: 16, r: 0.589
Predicted Capacity: 5.70, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.05, MCS Index: 9, M: 4, r: 0.478
Predicted Capacity: 7.37, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.59, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 13.33, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.87, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 4.14, MCS Index: 24, M: 64, r: 0.655
Predicted Capacity: 1.40, MCS Index: 11, M: 4, r: 0.617
Predicted Capacity: 13.77, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.46, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 3.76, MCS Index: 22, M: 16, r: 0.589
Predicted Capacity: 7.91, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 6.32, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.11, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.14, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.90, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.64, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 12.27, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 2.89, MCS Index: 19, M: 16, r: 0.678
Predicted Capacity: 4.07, MCS Index: 24, M: 64, r: 0.655
Predicted Capacity: 2.79, MCS Index: 19, M: 16, r: 0.678
Predicted Capacity: 11.67, MCS Index: 27, M: 64, r: 0.845
```

```
Predicted Capacity: 12.26, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.02, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 6.45, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 3.01, MCS Index: 20, M: 16, r: 0.736
Predicted Capacity: 9.42, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.36, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 2.55, MCS Index: 18, M: 16, r: 0.633
Predicted Capacity: 7.73, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 6.88, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 8.64, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 5.61, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 1.33, MCS Index: 11, M: 4, r: 0.617
Predicted Capacity: 7.44, MCS Index: 27, M: 64, r: 0.845
Predicted Capacity: 7.70, MCS Index: 27, M: 64, r: 0.845
```