```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from scipy.stats import zscore
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
import xgboost as xgb

df = pd.read_csv("dataset - waste forecasting set.csv")
df.head()
```

```
  iso3c region_id country_name income_id           gdp  \
0   ABW       LCN        Aruba       HIC  35563.312500
1   AFG       SAS  Afghanistan       LIC   2057.062256
2   AGO       SSF       Angola       LMC   8036.690430
3   ALB       ECS      Albania       UMC  13724.058590
4   AND       ECS      Andorra       HIC  43711.800780

   population_population_number_of_people  \
0                                  103187
1                                34656032
2                                25096150
3                                 2854191
4                                   82431

   total_msw_total_msw_generated_tons_year
0                             8.813202e+04
1                             5.628525e+06
2                             4.213644e+06
3                             1.087447e+06
4                             4.300000e+04
```

```python
df =pd.read_csv("country_level_data.csv")
df.head()
```

```
  iso3c region_id          country_name income_id          gdp  \
0   AGO       SSF                Angola       LMC   8036.69043
1   ALB       ECS               Albania       UMC  13724.05859
2   AND       ECS               Andorra       HIC  43711.80078
```

```
3   ARE       MEA  United Arab Emirates       HIC  67119.13281
4   ARG       LCN             Argentina       HIC  23550.09961

   population_population_number_of_people  \
0                                25096150
1                                 2854191
2                                   82431
3                                 9770529
4                                42981516

   total_msw_total_msw_generated_tons_year  \
0                             4.213644e+06
1                             1.087447e+06
2                             4.300000e+04
3                             5.617682e+06
4                             1.791055e+07

   composition_food_organic_waste_percent
composition_glass_percent  \
0                                   51.80                           6.70

1                                   51.40                           4.50

2                                   31.20                           8.20

3                                   39.00                           4.00

4                                   38.74                           3.16


   composition_metal_percent  composition_other_percent  \
0                       4.40                      11.50
1                       4.80                      15.21
2                       2.60                      11.60
3                       3.00                      10.00
4                       1.84                      15.36

   composition_paper_cardboard_percent  composition_plastic_percent
0                                11.90                        13.50
1                                 9.90                         9.60
2                                35.10                        11.30
3                                25.00                        19.00
4                                13.96                        14.61
```

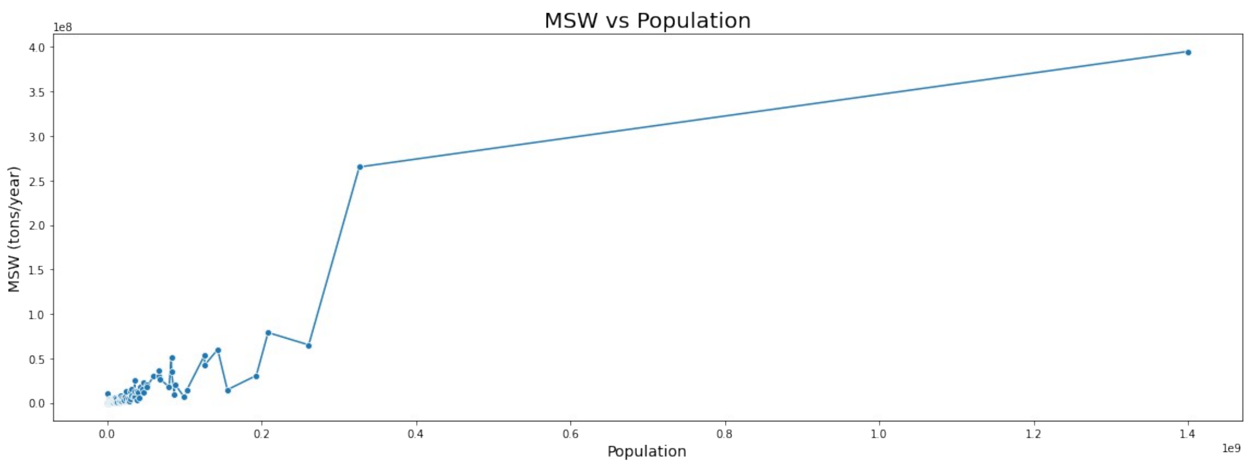
```
  income_id          gdp  population_population_number_of_people  \
0       LMC   8036.69043                                25096150   
1       UMC  13724.05859                                 2854191   
2       HIC  43711.80078                                   82431   
3       HIC  67119.13281                                 9770529   
4       HIC  23550.09961                                42981516   

   total_msw_total_msw_generated_tons_year  \
0                             4.213644e+06   
1                             1.087447e+06   
2                             4.300000e+04   
3                             5.617682e+06   
4                             1.791055e+07   

   composition_food_organic_waste_percent
composition_glass_percent  \
0                                   51.80                       6.70

1                                   51.40                       4.50

2                                   31.20                       8.20

3                                   39.00                       4.00

4                                   38.74                       3.16


   composition_metal_percent  composition_other_percent  \
0                       4.40                      11.50   
1                       4.80                      15.21   
2                       2.60                      11.60   
3                       3.00                      10.00   
4                       1.84                      15.36   

   composition_paper_cardboard_percent  composition_plastic_percent
0                                11.90                        13.50
1                                 9.90                         9.60
2                                35.10                        11.30
3                                25.00                        19.00
4                                13.96                        14.61
```

```
df = df.iloc[:, 3:]
df.head()
```

```
  income_id          gdp  population_population_number_of_people  \
0       LMC   8036.69043                                25096150   
1       UMC  13724.05859                                 2854191   
2       HIC  43711.80078                                   82431   
3       HIC  67119.13281                                 9770529   
4       HIC  23550.09961                                42981516   
```

```
   total_msw_total_msw_generated_tons_year  \
0                             4.213644e+06
1                             1.087447e+06
2                             4.300000e+04
3                             5.617682e+06
4                             1.791055e+07

   composition_food_organic_waste_percent
composition_glass_percent  \
0                                    51.80                          6.70

1                                    51.40                          4.50

2                                    31.20                          8.20

3                                    39.00                          4.00

4                                    38.74                          3.16


   composition_metal_percent  composition_other_percent  \
0                       4.40                      11.50
1                       4.80                      15.21
2                       2.60                      11.60
3                       3.00                      10.00
4                       1.84                      15.36

   composition_paper_cardboard_percent  composition_plastic_percent
0                                11.90                        13.50
1                                 9.90                         9.60
2                                35.10                        11.30
3                                25.00                        19.00
4                                13.96                        14.61

df.rename(columns={
    'population_population_number_of_people': 'population',
    'total_msw_total_msw_generated_tons_year': 'msw',
    'gdp': 'gdp',
    'income_id': 'income_id'
}, inplace=True)

df['population'] = pd.to_numeric(df['population'], errors='coerce')
df['gdp'] = pd.to_numeric(df['gdp'], errors='coerce')
df['msw'] = pd.to_numeric(df['msw'], errors='coerce')
df_clean = df.dropna(subset=['population', 'gdp', 'msw', 'income_id'])

income_grouped = df_clean.groupby('income_id')
['msw'].mean().reset_index()

plt.figure(figsize=(16, 6))
df_sorted_pop = df_clean.sort_values(by='population')
```

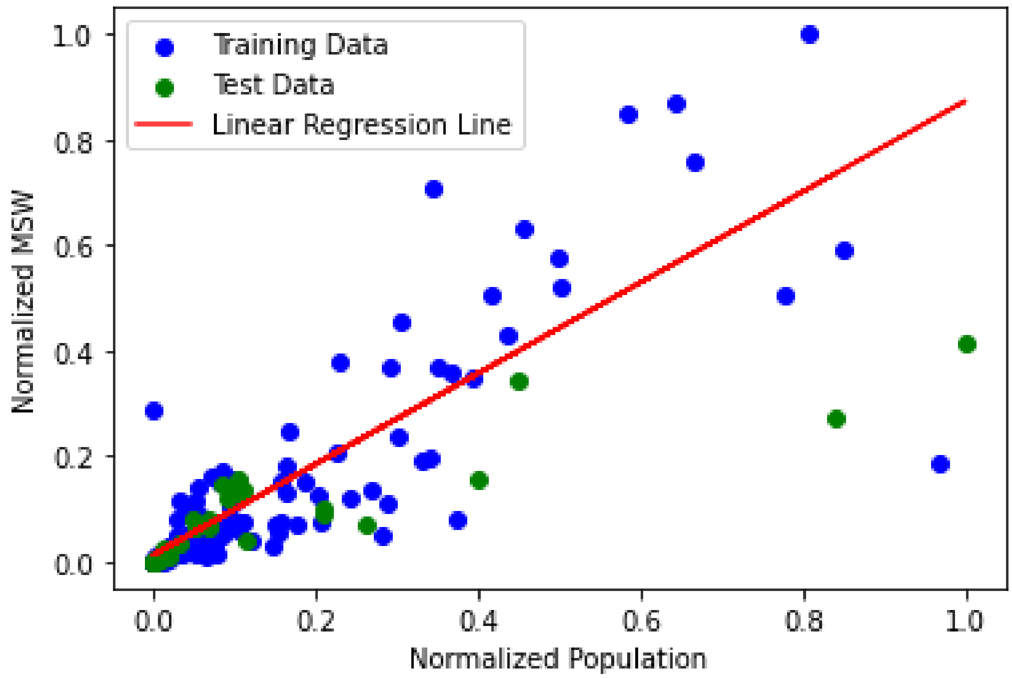
```
sns.lineplot(
    x='population',
    y='msw',
    data=df_sorted_pop,
    marker='o'
)
plt.title('MSW vs Population', fontsize=20)
plt.xlabel('Population', fontsize=14)
plt.ylabel('MSW (tons/year)', fontsize=14)
plt.tight_layout()
plt.show()
```




```
population_threshold = df['population'].quantile(0.95)  # 95th
percentile as an upper bound
msw_threshold = df['msw'].quantile(0.95)  # 95th percentile as an
upper bound
df_filtered = df[(df['population'] <= population_threshold) &
                 (df['msw'] <= msw_threshold)]

scaler = MinMaxScaler()
df_filtered[['population', 'msw']] = scaler.fit_transform(
    df_filtered[['population', 'msw']])

X = df_filtered[['population']]  # Independent variable
y = df_filtered['msw']  # Dependent variable

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

model1 = LinearRegression()
model2 = RandomForestRegressor(random_state=42)
model3 = xgb.XGBRegressor(random_state=42)
model4 = MLPRegressor(random_state=42,max_iter=1000)
```

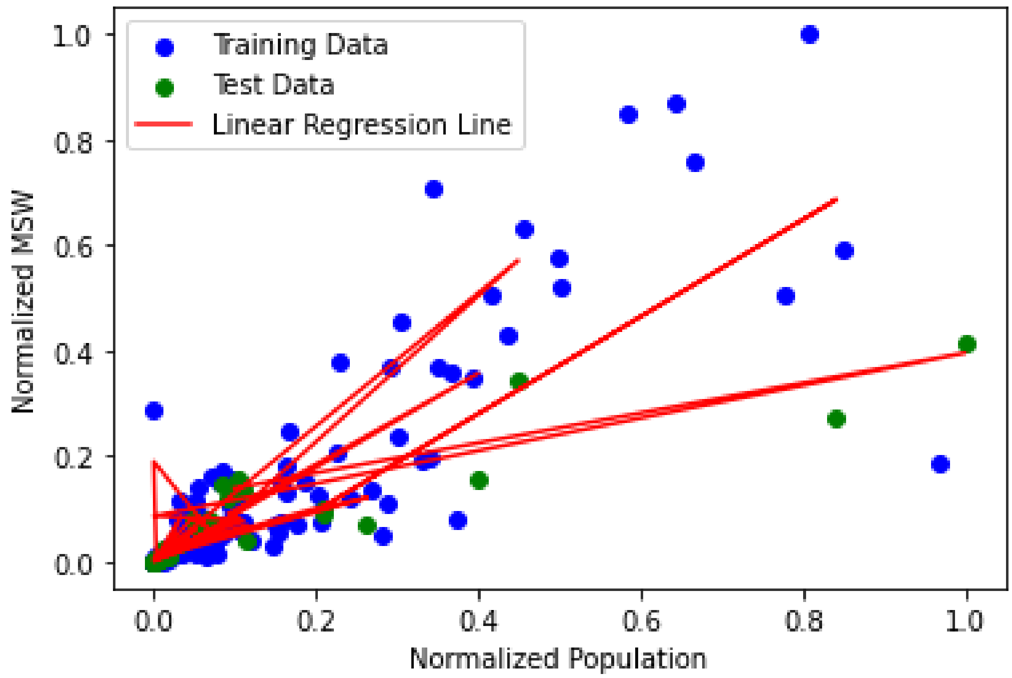
```python
model1.fit(X_train, y_train)
y1_pred = model1.predict(X_test)
model2.fit(X_train, y_train)
y2_pred = model2.predict(X_test)
model3.fit(X_train, y_train)
y3_pred = model3.predict(X_test)
model4.fit(X_train, y_train)
y4_pred = model4.predict(X_test)

plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y1_pred, color='red', label='Linear Regression Line')
plt.xlabel('Normalized Population')
plt.ylabel('Normalized MSW')
plt.legend()
plt.show()


mse = mean_squared_error(y_test, y1_pred)
r2 = r2_score(y_test, y1_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
/var/folders/93/q7yjwcrd1hg2r5l0md2nggvw0000gn/T/
ipykernel_1949/3625724015.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_filtered[['population', 'msw']] = scaler.fit_transform(
```



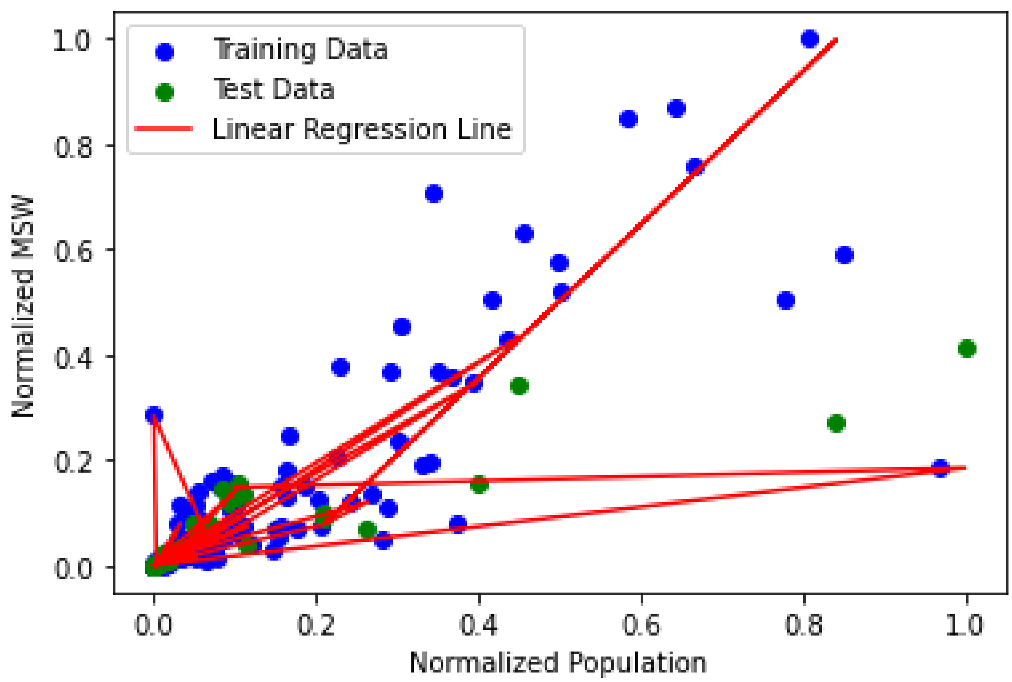

```
Mean Squared Error: 0.01624367206570611
R-squared: -0.639236866904294
```

```
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y2_pred, color='red', label='Linear Regression Line')
plt.xlabel('Normalized Population')
plt.ylabel('Normalized MSW')
plt.legend()
plt.show()


mse = mean_squared_error(y_test, y2_pred)
r2 = r2_score(y_test, y2_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

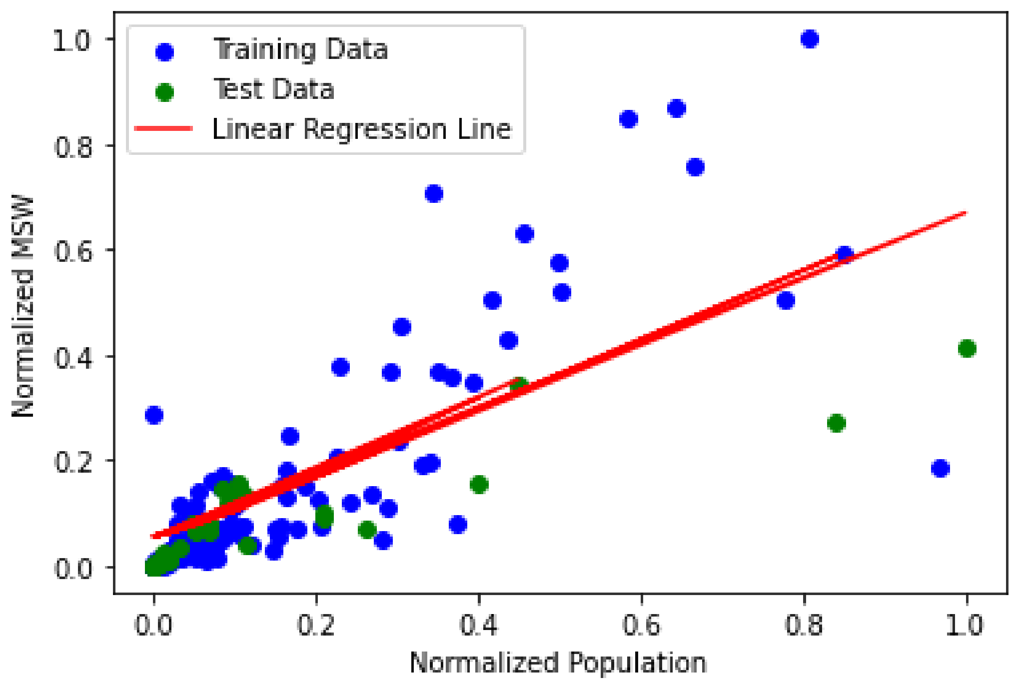



```
Mean Squared Error: 0.010079376119244922
R-squared: -0.017164398741066123
```

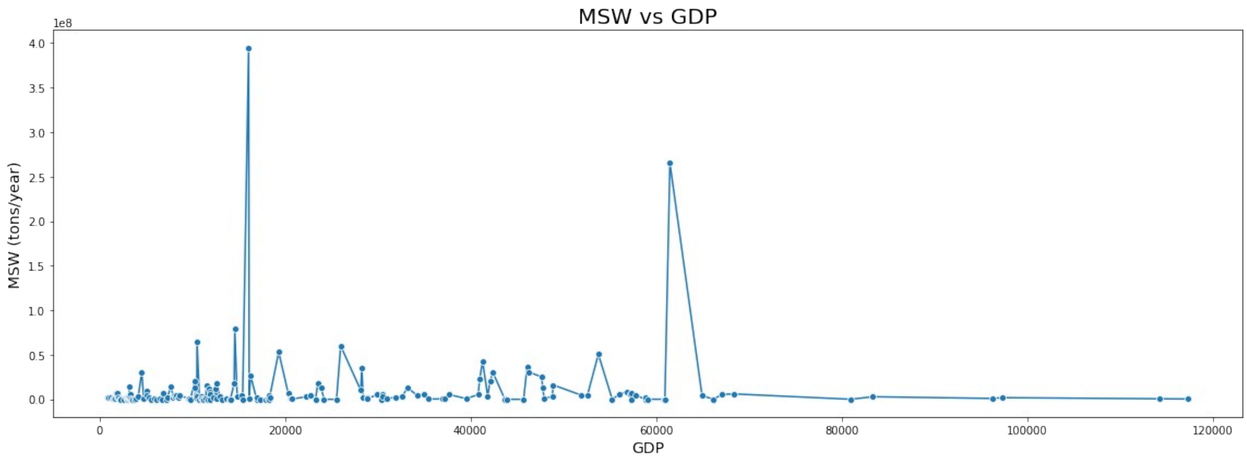
```
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y3_pred, color='red', label='Linear Regression Line')
plt.xlabel('Normalized Population')
plt.ylabel('Normalized MSW')
plt.legend()
plt.show()


mse = mean_squared_error(y_test, y3_pred)
r2 = r2_score(y_test, y3_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```




```
Mean Squared Error: 0.02221996497800744
R-squared: -1.2423369313254309

plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y4_pred, color='red', label='Linear Regression Line')
plt.xlabel('Normalized Population')
plt.ylabel('Normalized MSW')
plt.legend()
plt.show()


mse = mean_squared_error(y_test, y4_pred)
r2 = r2_score(y_test, y4_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```



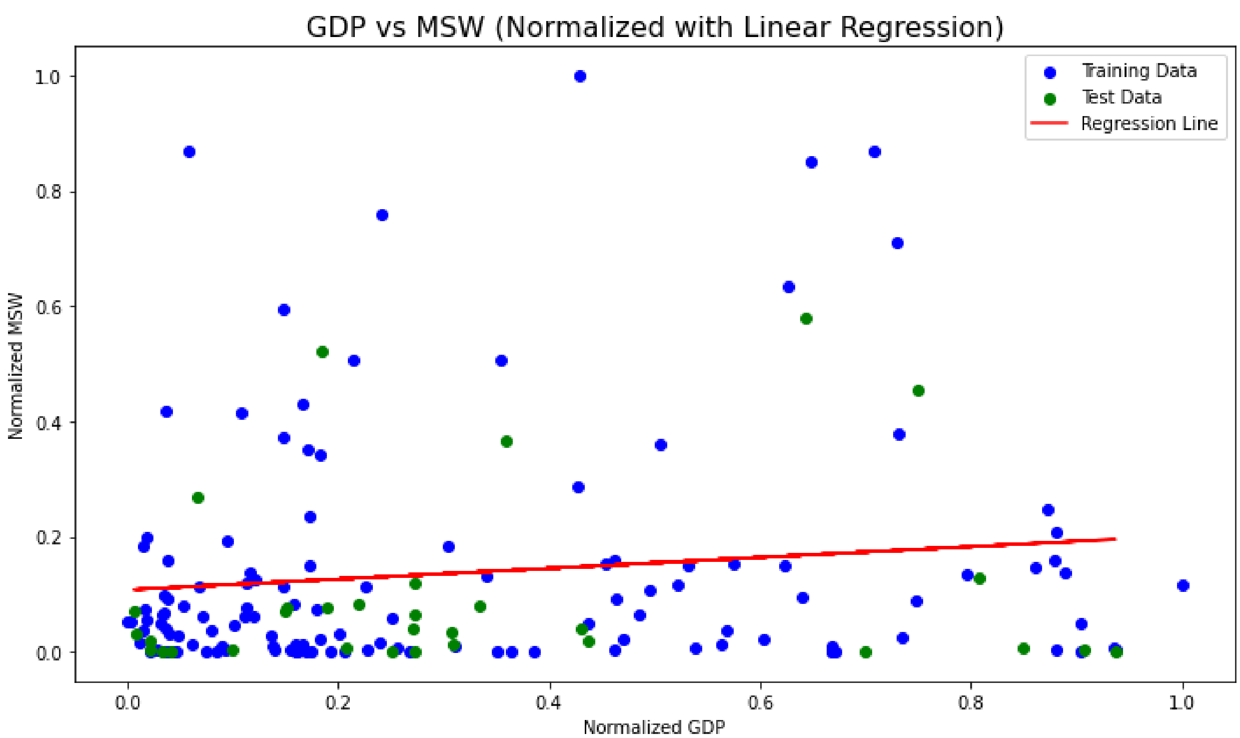

```
Mean Squared Error: 0.008439052928901122
R-squared: 0.14836949263355892

# Plot 2: GDP vs MSW
plt.figure(figsize=(16, 6))
df_sorted_gdp = df_clean.sort_values(by='gdp')
sns.lineplot(
    x='gdp',
    y='msw',
    data=df_sorted_gdp,
    marker='o'
)
plt.title('MSW vs GDP', fontsize=20)
plt.xlabel('GDP', fontsize=14)
plt.ylabel('MSW (tons/year)', fontsize=14)
plt.tight_layout()
plt.show()
```




```
gdp_threshold = df['gdp'].quantile(0.95)
msw_threshold = df['msw'].quantile(0.95)
df_filtered = df[(df['gdp'] <= gdp_threshold) & (df['msw'] <=
msw_threshold)]
# Normalize using MinMaxScaler
scaler = MinMaxScaler()
df_filtered[['gdp', 'msw']] = scaler.fit_transform(df_filtered[['gdp',
'msw']])
# Prepare features and target
X = df_filtered[['gdp']]
y = df_filtered['msw']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Linear regression
model = LinearRegression()
model2 = RandomForestRegressor(random_state=42)
model3 = xgb.XGBRegressor(random_state=42)
model4 = MLPRegressor(random_state=42,max_iter=1000)
model.fit(X_train, y_train)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)
model4.fit(X_train, y_train)

y1_pred = model.predict(X_test)
y2_pred = model2.predict(X_test)
y3_pred = model3.predict(X_test)
y4_pred = model4.predict(X_test)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y1_pred, color='red', label='Regression Line')
plt.xlabel('Normalized GDP')
```

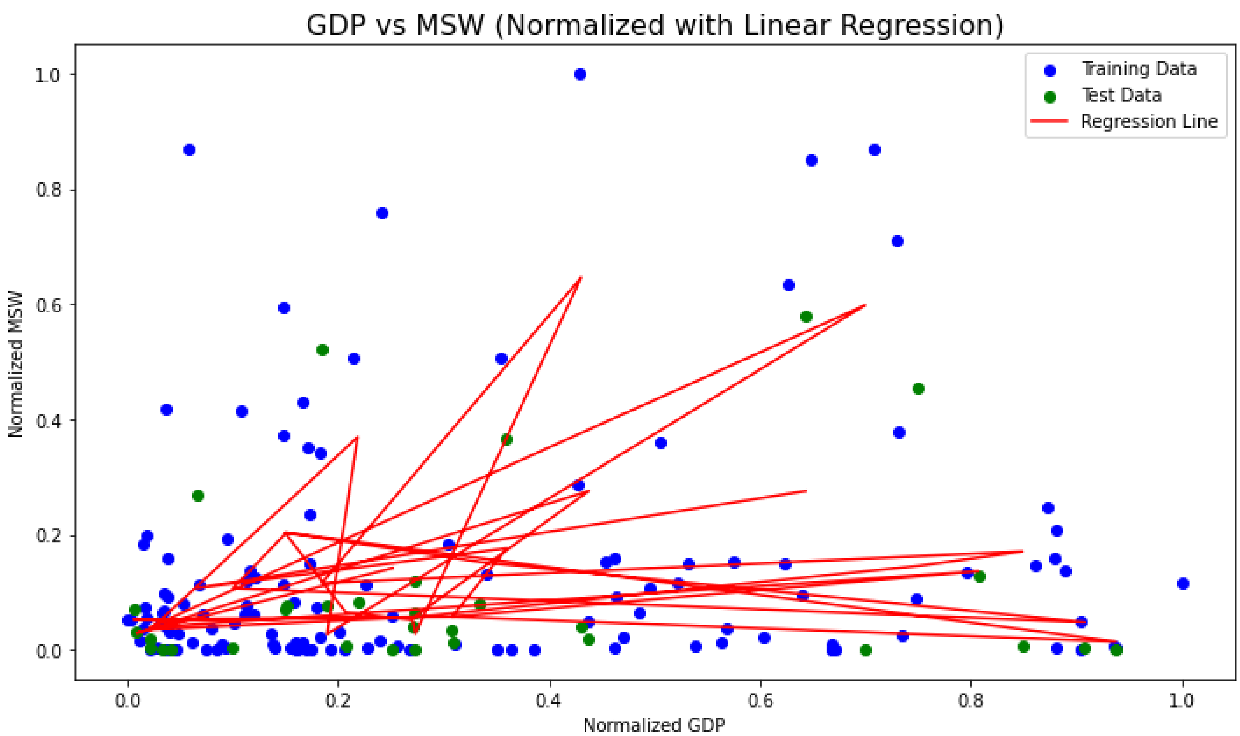
```
plt.ylabel('Normalized MSW')
plt.title('GDP vs MSW (Normalized with Linear Regression)',
fontsize=16)
plt.legend()
plt.tight_layout()
plt.show()

# Evaluation
mse = mean_squared_error(y_test, y1_pred)
r2 = r2_score(y_test, y1_pred)
print(f'Mean Squared Error: {mse:.4f}')
print(f'R-squared: {r2:.4f}')
```

```
/var/folders/93/q7yjwcrd1hg2r5l0md2nggvw0000gn/T/
ipykernel_1949/2438894860.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_filtered[['gdp', 'msw']] =
scaler.fit_transform(df_filtered[['gdp', 'msw']])
```




```
Mean Squared Error: 0.0253
R-squared: -0.0380
```

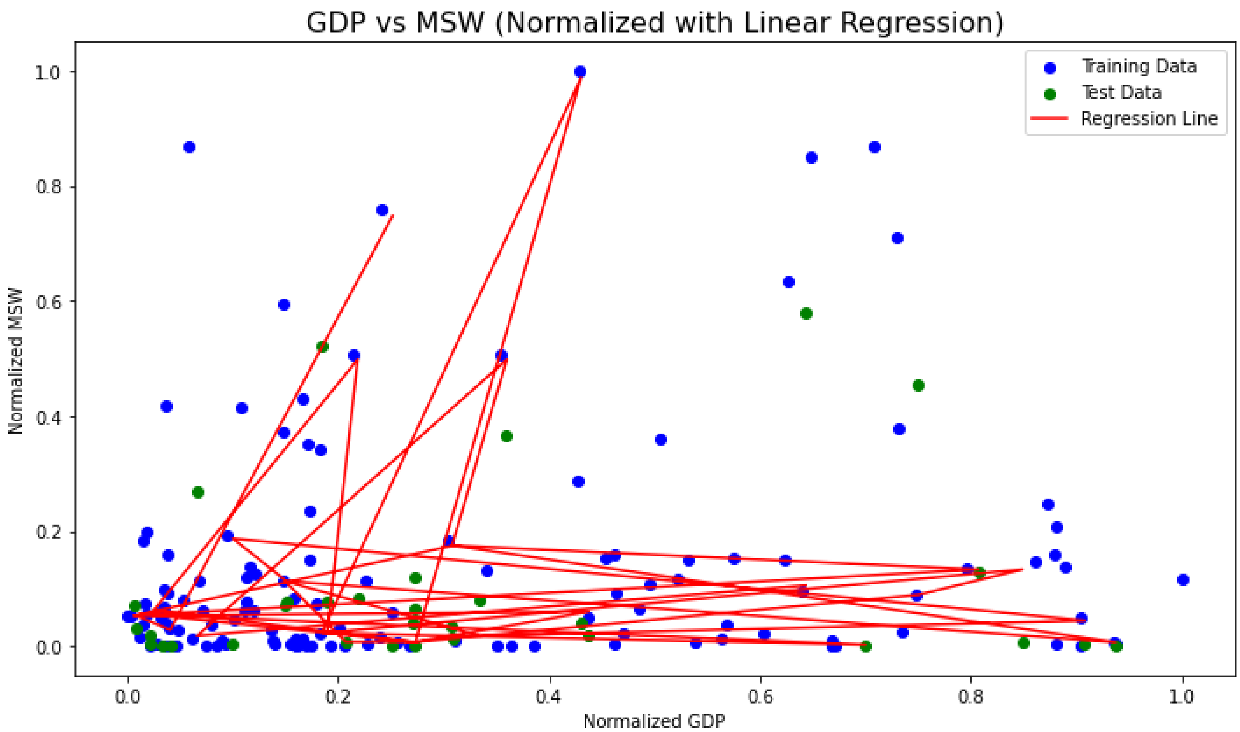
```
# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y2_pred, color='red', label='Regression Line')
plt.xlabel('Normalized GDP')
plt.ylabel('Normalized MSW')
plt.title('GDP vs MSW (Normalized with Linear Regression)',
fontsize=16)
plt.legend()
plt.tight_layout()
plt.show()

# Evaluation
mse = mean_squared_error(y_test, y2_pred)
r2 = r2_score(y_test, y2_pred)
print(f'Mean Squared Error: {mse:.4f}')
print(f'R-squared: {r2:.4f}')
```




```
Mean Squared Error: 0.0439
R-squared: -0.8034
```

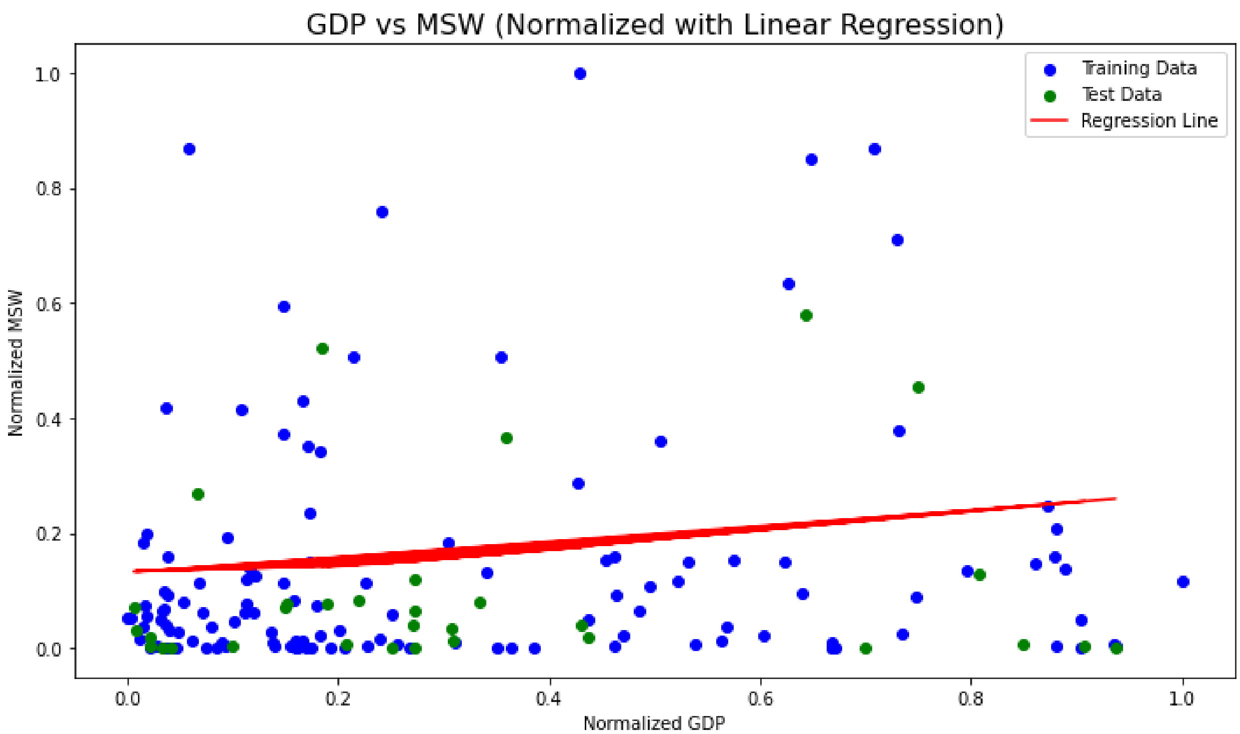
```
# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y3_pred, color='red', label='Regression Line')
```

```
plt.xlabel('Normalized GDP')
plt.ylabel('Normalized MSW')
plt.title('GDP vs MSW (Normalized with Linear Regression)',
fontsize=16)
plt.legend()
plt.tight_layout()
plt.show()

# Evaluation
mse = mean_squared_error(y_test, y3_pred)
r2 = r2_score(y_test, y3_pred)
print(f'Mean Squared Error: {mse:.4f}')
print(f'R-squared: {r2:.4f}')
```

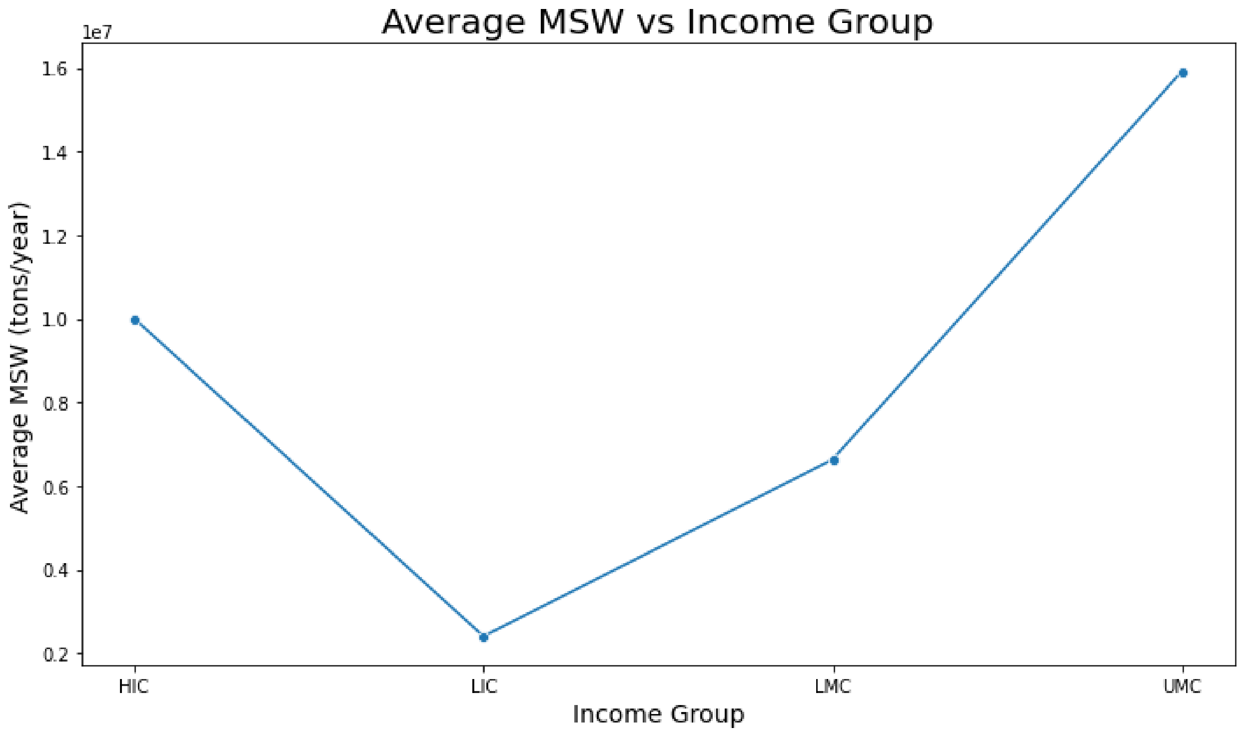



```
Mean Squared Error: 0.0764
R-squared: -2.1371

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y4_pred, color='red', label='Regression Line')
plt.xlabel('Normalized GDP')
plt.ylabel('Normalized MSW')
plt.title('GDP vs MSW (Normalized with Linear Regression)',
fontsize=16)
plt.legend()
```

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

# Evaluation
mse = mean_squared_error(y_test, y4_pred)
r2 = r2_score(y_test, y4_pred)
print(f'Mean Squared Error: {mse:.4f}')
print(f'R-squared: {r2:.4f}')
```



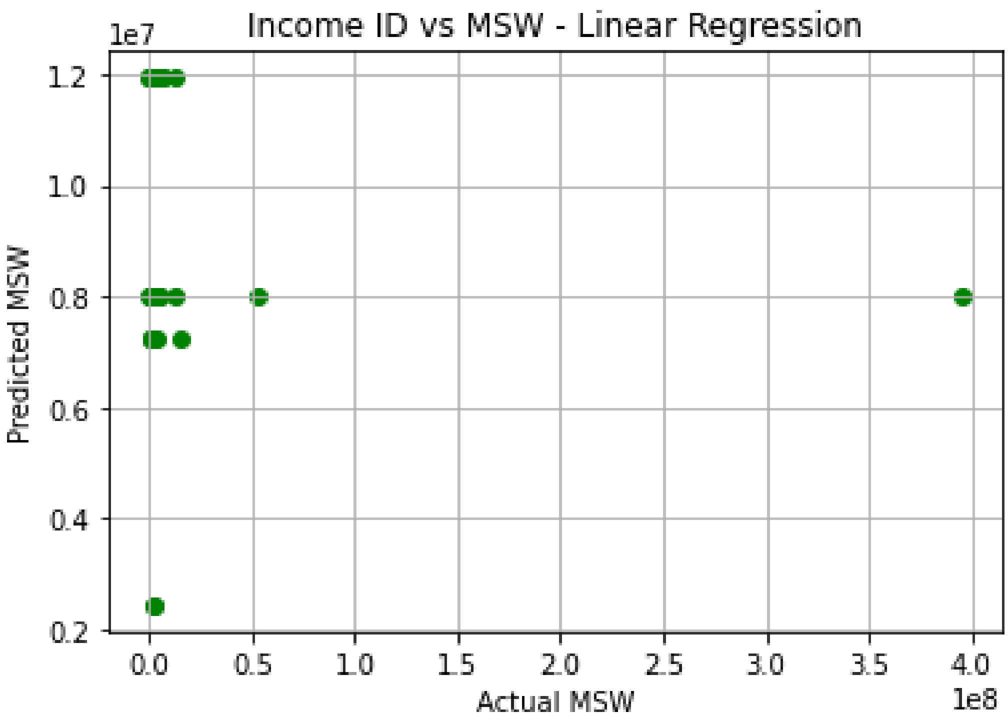

```
Mean Squared Error: 0.0288
R-squared: -0.1804

plt.figure(figsize=(10, 6))
sns.lineplot(
    x='income_id',
    y='msw',
    data=income_grouped,
    marker='o'
)
plt.title('Average MSW vs Income Group', fontsize=20)
plt.xlabel('Income Group', fontsize=14)
plt.ylabel('Average MSW (tons/year)', fontsize=14)
plt.tight_layout()
plt.show()
```




```
# df_clean = df[df['msw'] < df['msw'].quantile(0.95)]

df_encoded = pd.get_dummies(df, columns=['income_id'],
drop_first=True)
X = df_encoded[['income_id_LIC', 'income_id_LMC' , 'income_id_UMC']]
# 'income_id_HIC' is dropped (baseline)
y = df_encoded['msw']

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

# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluation metrics
mse_score = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse_score)
print("R-squared:", r2)

plt.scatter(y_test, y_pred, color='green')
```

```
plt.xlabel('Actual MSW')
plt.ylabel('Predicted MSW')
plt.title('Income ID vs MSW - Linear Regression')
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 4391429448605500.5
R-squared: -0.02021739618154461
```

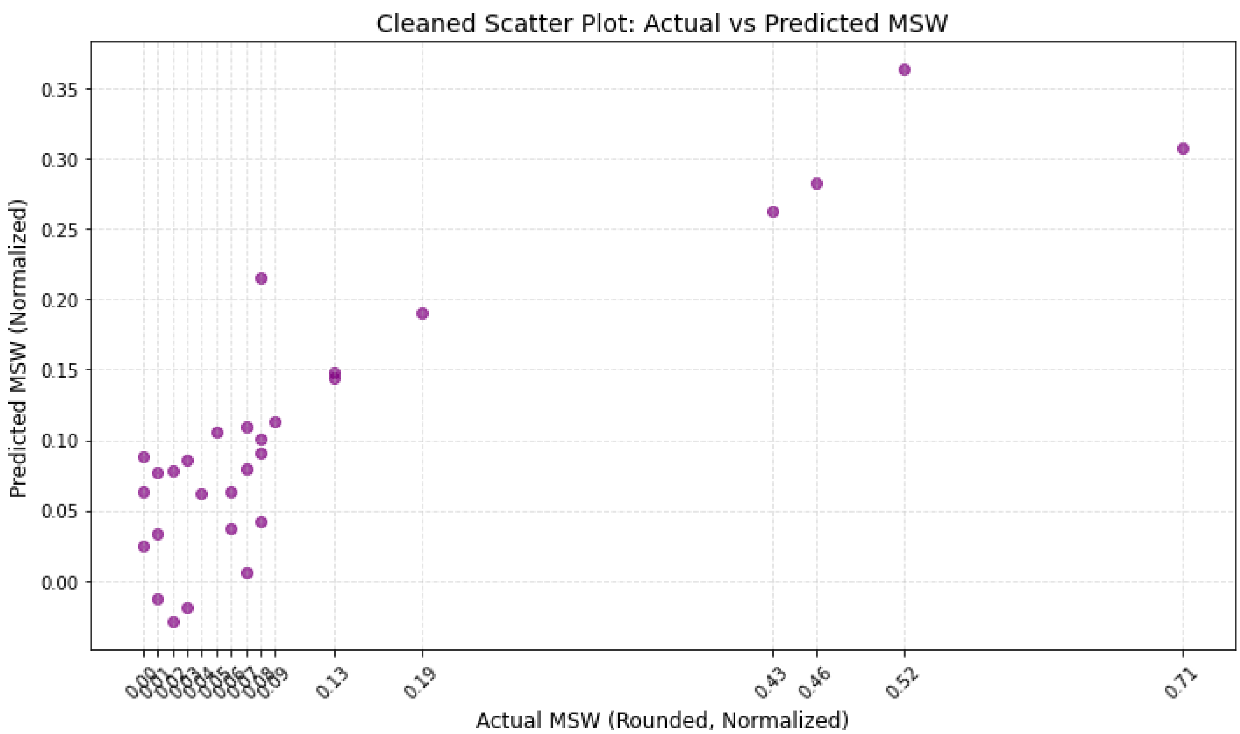



```
# Step 1: Filter extreme values (95th percentile)
population_threshold = df['population'].quantile(0.95)
gdp_threshold = df['gdp'].quantile(0.95)
msw_threshold = df['msw'].quantile(0.95)

df_filtered = df[(df['population'] <= population_threshold) &
                 (df['gdp'] <= gdp_threshold) &
                 (df['msw'] <= msw_threshold)]

# Step 2: One-hot encode income groups
df_encoded = pd.get_dummies(df_filtered, columns=['income_id'],
drop_first=True)

# Step 3: Normalize numerical features
scaler = MinMaxScaler()
```

```python
df_encoded[['population', 'gdp', 'msw']] = scaler.fit_transform(
    df_encoded[['population', 'gdp', 'msw']]
)

# Step 4: Define independent variables and target
feature_columns = ['population', 'gdp', 'income_id_LIC',
'income_id_LMC', 'income_id_UMC']
X = df_encoded[feature_columns]
y = df_encoded['msw']

# Step 5: Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 6: Train the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
model2 = RandomForestRegressor(random_state=42)
model3 = xgb.XGBRegressor(random_state=42)
model4 = MLPRegressor(random_state=42,max_iter=1000)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)
model4.fit(X_train, y_train)

# Step 7: Predict
y1_pred = model.predict(X_test)
y2_pred = model2.predict(X_test)
y3_pred = model3.predict(X_test)
y4_pred = model4.predict(X_test)


# Step 8: Evaluate
mse = mean_squared_error(y_test, y1_pred)
r2 = r2_score(y_test, y1_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")

threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y1_pred_filtered = y1_pred[filtered_indices]

# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y1_pred_filtered[sorted_indices]

# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)
```

```
plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

```
Mean Squared Error: 0.0196
R-squared: 0.7130
```

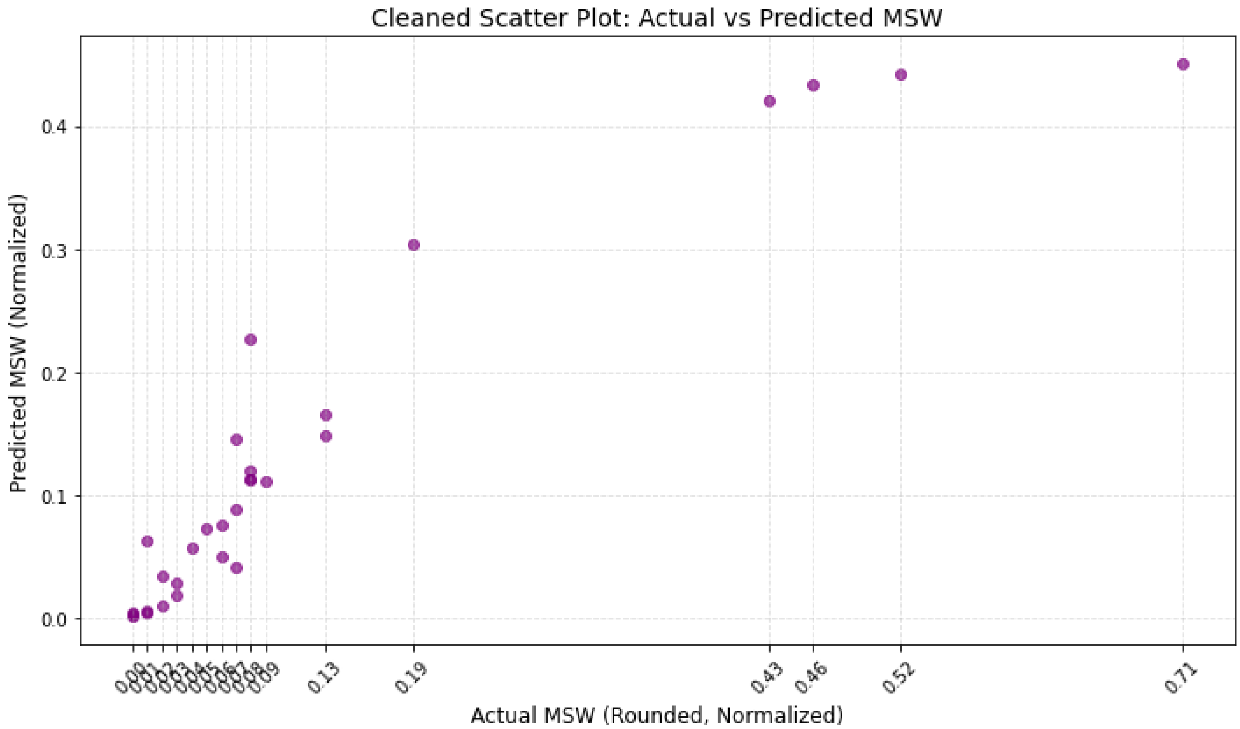



```
# Check the shapes of y_test and y_pred
print(f"Shape of y_test: {y_test.shape}")
print(f"Shape of y_pred: {y4_pred.shape}")
```

```
Shape of y_test: (31,)
Shape of y_pred: (31,)
```

```python
# Step 8: Evaluate
mse = mean_squared_error(y_test, y2_pred)
r2 = r2_score(y_test, y2_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")

threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y2_pred_filtered = y2_pred[filtered_indices]

# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y2_pred_filtered[sorted_indices]

# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

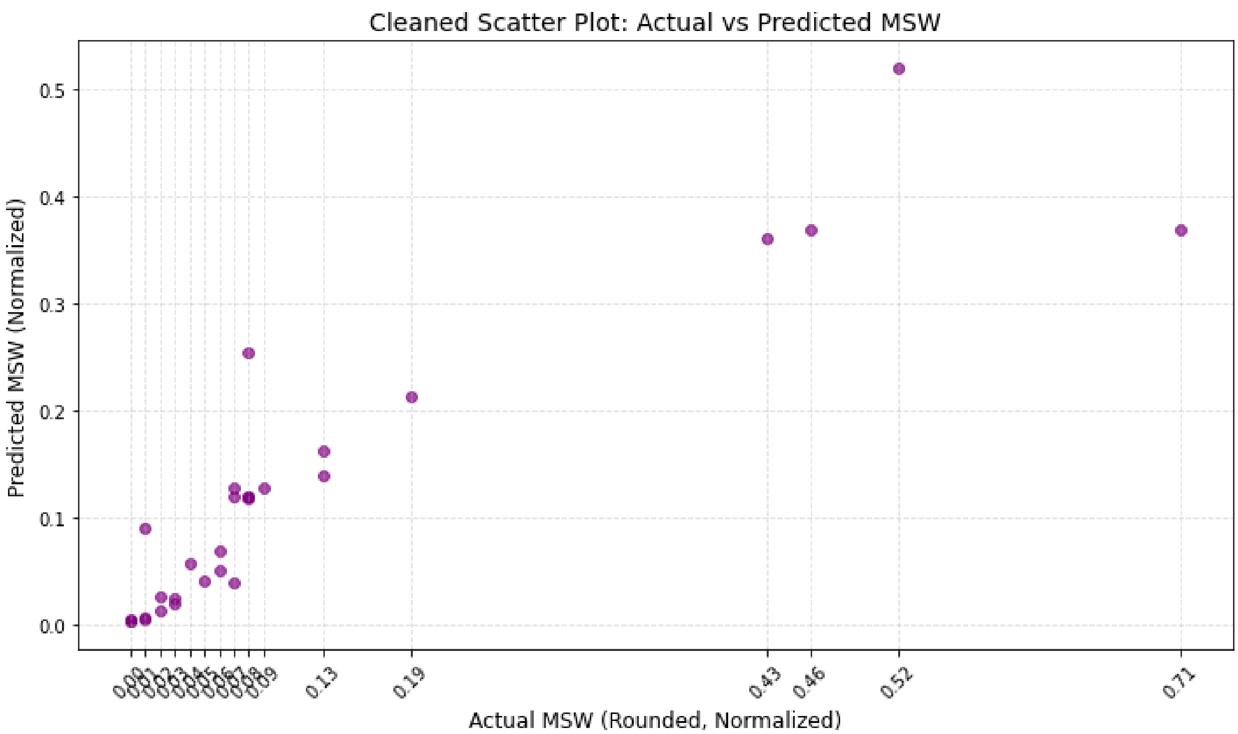
```
Mean Squared Error: 0.0100
R-squared: 0.8530
```




```
# Step 8: Evaluate
mse = mean_squared_error(y_test, y3_pred)
r2 = r2_score(y_test, y3_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")

threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y3_pred_filtered = y3_pred[filtered_indices]
# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y3_pred_filtered[sorted_indices]

# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)
```

```
# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

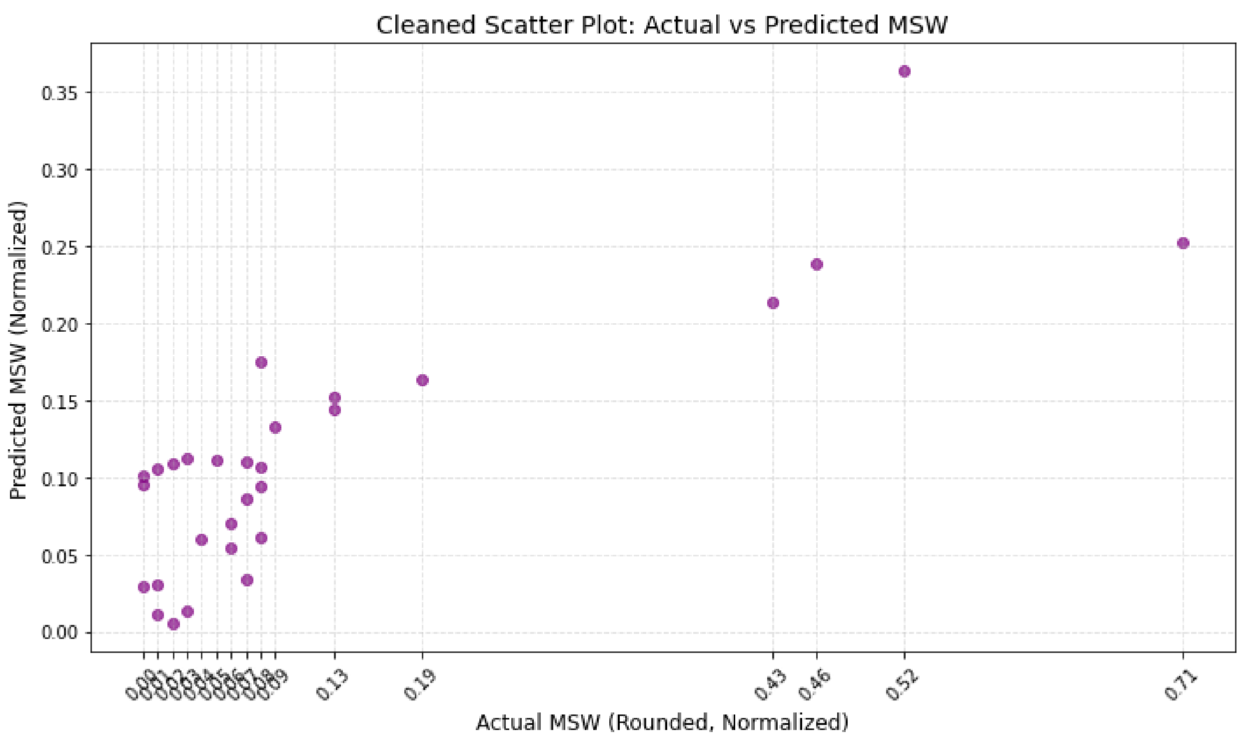
```
Mean Squared Error: 0.0086
R-squared: 0.8737
```




```
# Step 8: Evaluate
mse = mean_squared_error(y_test, y4_pred)
r2 = r2_score(y_test, y4_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")

threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y4_pred_filtered = y4_pred[filtered_indices]
# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y4_pred_filtered[sorted_indices]
```

```python
# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

```
Mean Squared Error: 0.0282
R-squared: 0.5876
```

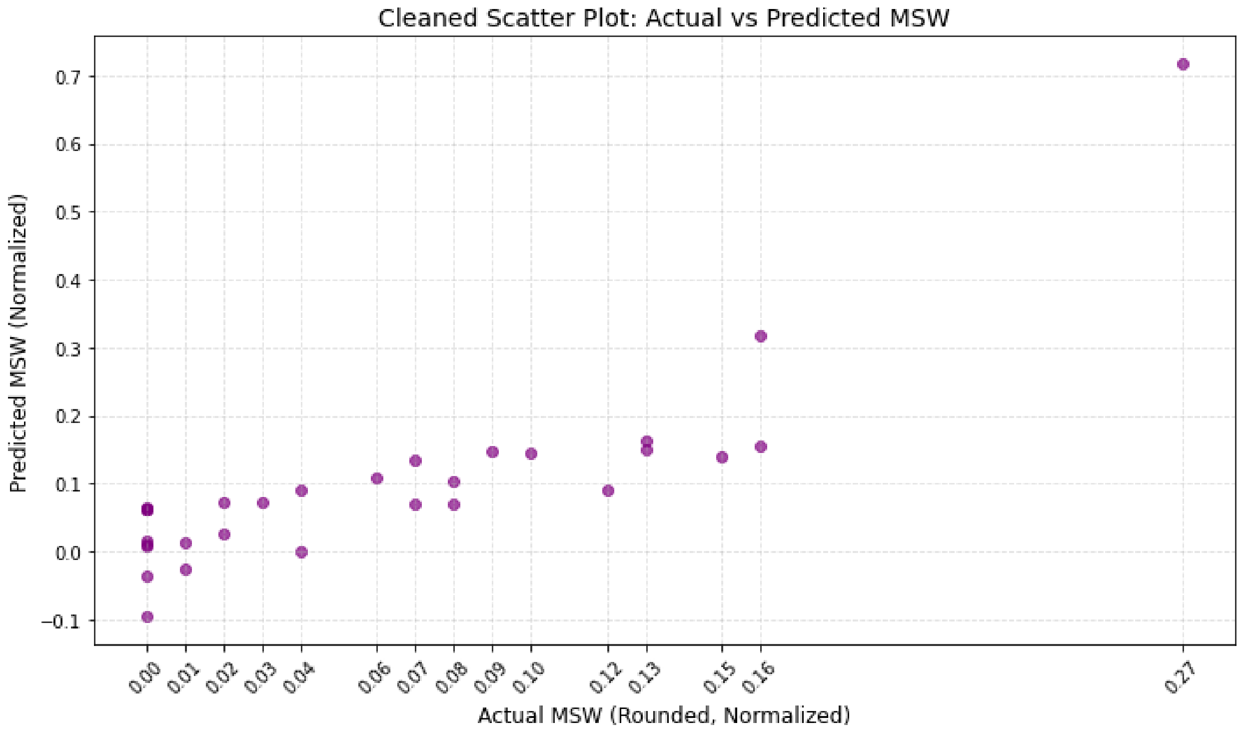



```python
#REMOVING GDP
# Step 1: Filter extreme values (95th percentile)
population_threshold = df['population'].quantile(0.95)
```

```
msw_threshold = df['msw'].quantile(0.95)

df_filtered = df[(df['population'] <= population_threshold) &

                 (df['msw'] <= msw_threshold)]

# Step 2: One-hot encode income groups
df_encoded = pd.get_dummies(df_filtered, columns=['income_id'],
drop_first=True)

# Step 3: Normalize numerical features
scaler = MinMaxScaler()
df_encoded[['population', 'msw']] = scaler.fit_transform(
    df_encoded[['population', 'msw']]
)

# Step 4: Define independent variables and target
feature_columns = ['population', 'income_id_LIC', 'income_id_LMC',
'income_id_UMC']
X = df_encoded[feature_columns]
y = df_encoded['msw']

# Step 5: Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 6: Train the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
model2 = RandomForestRegressor(random_state=42)
model3 = xgb.XGBRegressor(random_state=42)
model4 = MLPRegressor(random_state=42,max_iter=1000)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)
model4.fit(X_train, y_train)

# Step 7: Predict
y_pred = model.predict(X_test)
y2_pred = model2.predict(X_test)
y3_pred = model3.predict(X_test)
y4_pred = model4.predict(X_test)

# Step 8: Evaluate
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")
```

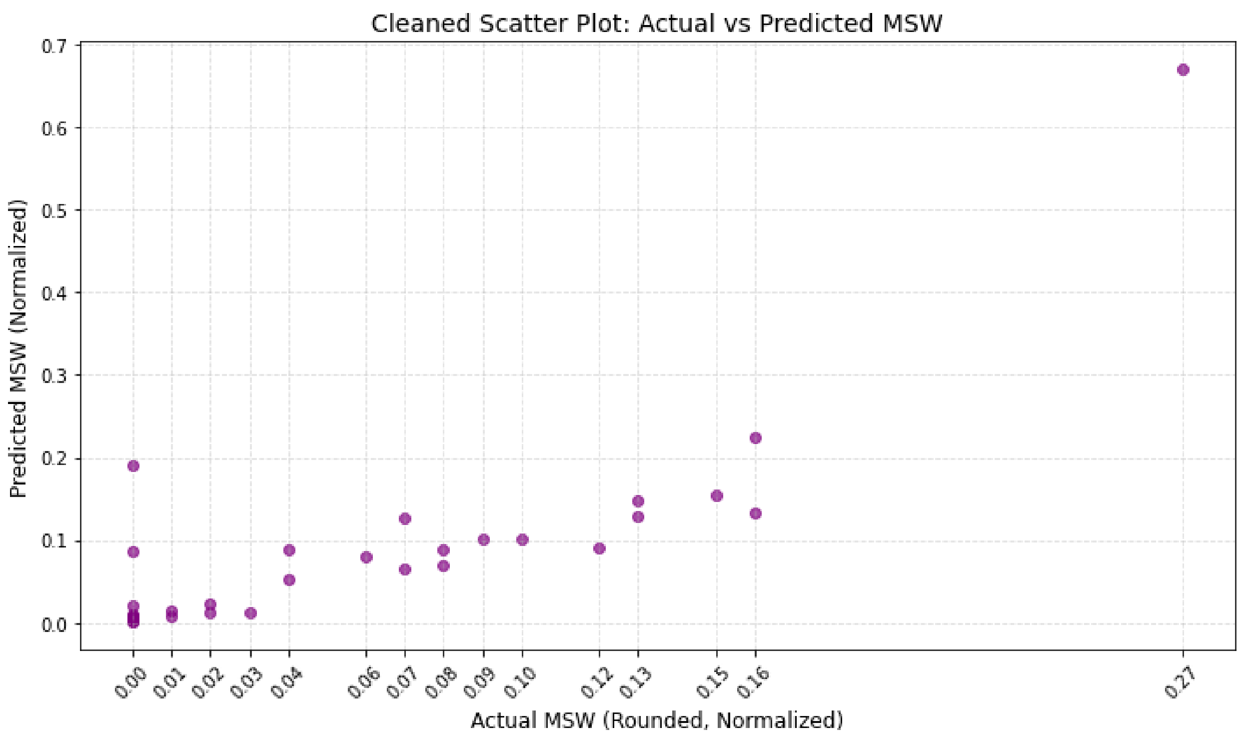
```python
threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y_pred[filtered_indices]

# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

```
Mean Squared Error: 0.0147
R-squared: -0.4818
```



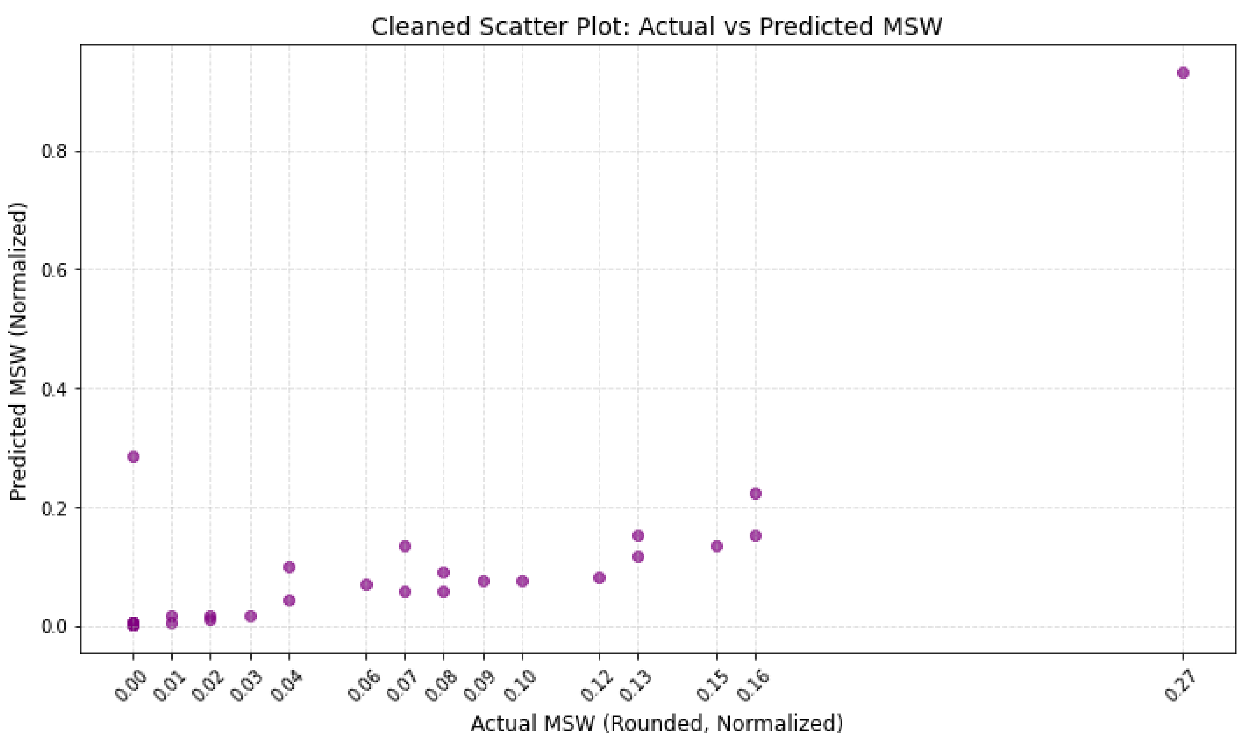

```
# Step 8: Evaluate
mse = mean_squared_error(y_test, y2_pred)
r2 = r2_score(y_test, y2_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")

threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y2_pred[filtered_indices]

# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
```

```
fontsize=14)

# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()

Mean Squared Error: 0.0082
R-squared: 0.1769
```



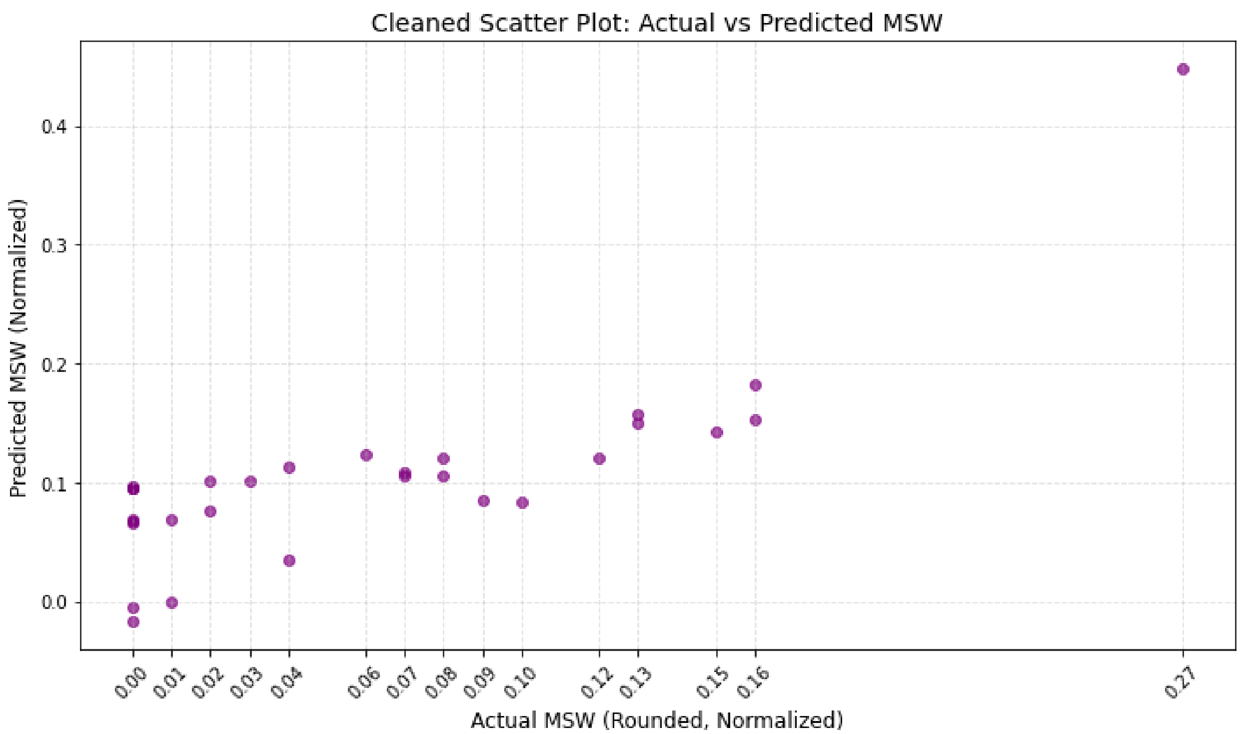

```
# Step 8: Evaluate
mse = mean_squared_error(y_test, y3_pred)
r2 = r2_score(y_test, y3_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")

threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y3_pred[filtered_indices]

# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
```

```
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()

Mean Squared Error: 0.0181
R-squared: -0.8220
```



```python
# Step 8: Evaluate
mse = mean_squared_error(y_test, y4_pred)
r2 = r2_score(y_test, y4_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared: {r2:.4f}")

threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y4_pred[filtered_indices]

# Sort for smoother visual appearance
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x values to reduce clutter (e.g., to 2 decimals)
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Use unique x values as ticks with spacing
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

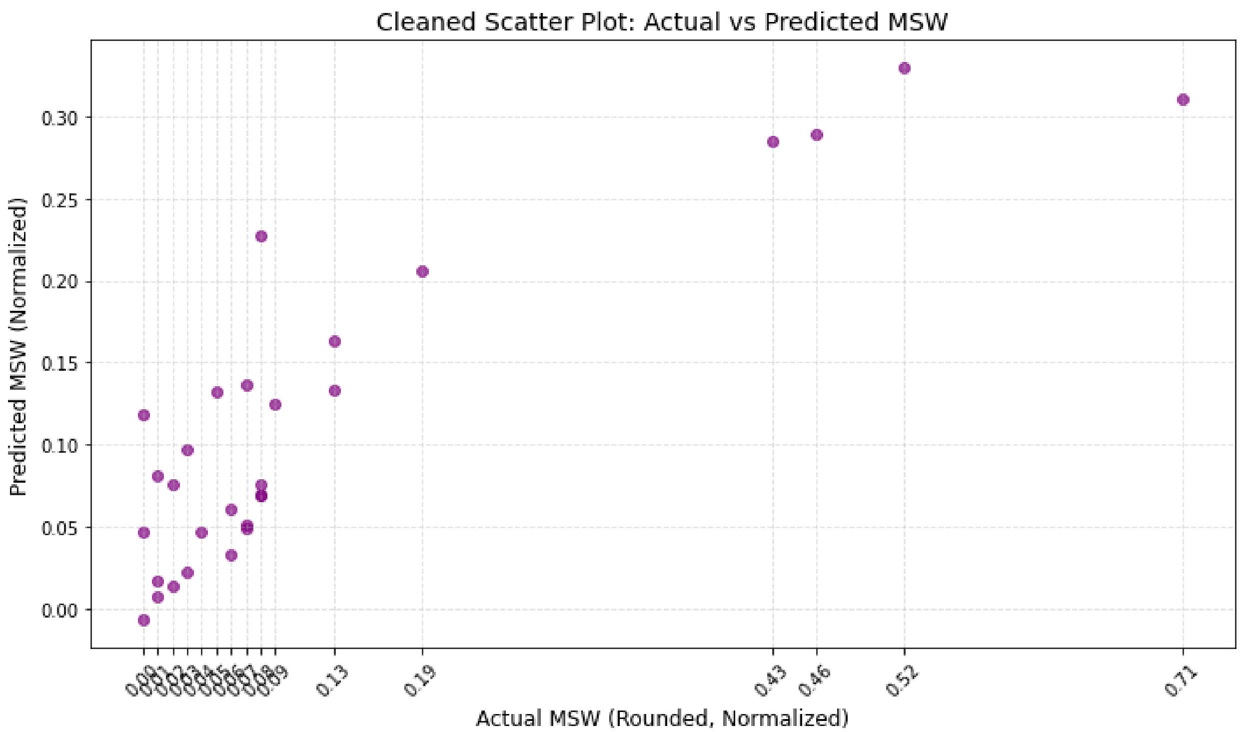
```
Mean Squared Error: 0.0042
R-squared: 0.5766
```




```python
#REMOVING INCOME_ID
# Step 1: Filter extreme values (95th percentile)
population_threshold = df['population'].quantile(0.95)
gdp_threshold = df['gdp'].quantile(0.95)
msw_threshold = df['msw'].quantile(0.95)

df_filtered = df[(df['population'] <= population_threshold) &
                 (df['gdp'] <= gdp_threshold) &
                 (df['msw'] <= msw_threshold)]

# Step 2: Normalize the numerical columns
scaler = MinMaxScaler()
df_filtered[['population', 'gdp', 'msw']] =
scaler.fit_transform(df_filtered[['population', 'gdp', 'msw']])

# Step 3: Define X and y
X = df_filtered[['population', 'gdp']]
y = df_filtered['msw']

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 6: Train the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
model2 = RandomForestRegressor(random_state=42)
```

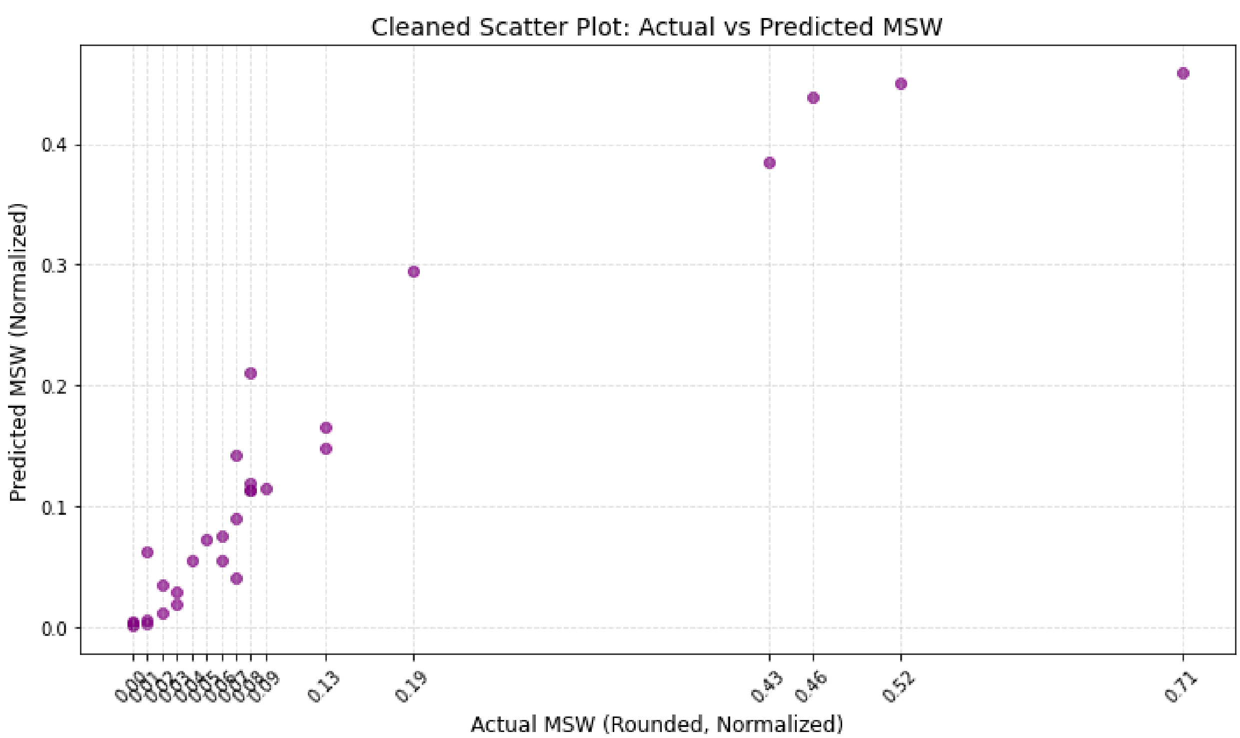
```python
model3 = xgb.XGBRegressor(random_state=42)
model4 = MLPRegressor(random_state=42,max_iter=1000)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)
model4.fit(X_train, y_train)

# Step 7: Predict
y_pred = model.predict(X_test)
y2_pred = model2.predict(X_test)
y3_pred = model3.predict(X_test)
y4_pred = model4.predict(X_test)

# Step 7: Evaluate
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.6f}")
print(f"R-squared: {r2:.6f}")

# Step 8: Graph – Cleaned Scatter Plot (95% threshold)
threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y_pred[filtered_indices]

# Sort for smoother plotting
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x-axis values to reduce clutter
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Unique x values for ticks
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

```
/var/folders/93/q7yjwcrd1hg2r5l0md2nggvw0000gn/T/
ipykernel_1949/1873524904.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_filtered[['population', 'gdp', 'msw']] =
scaler.fit_transform(df_filtered[['population', 'gdp', 'msw']])

Mean Squared Error: 0.020429
R-squared: 0.700827
```



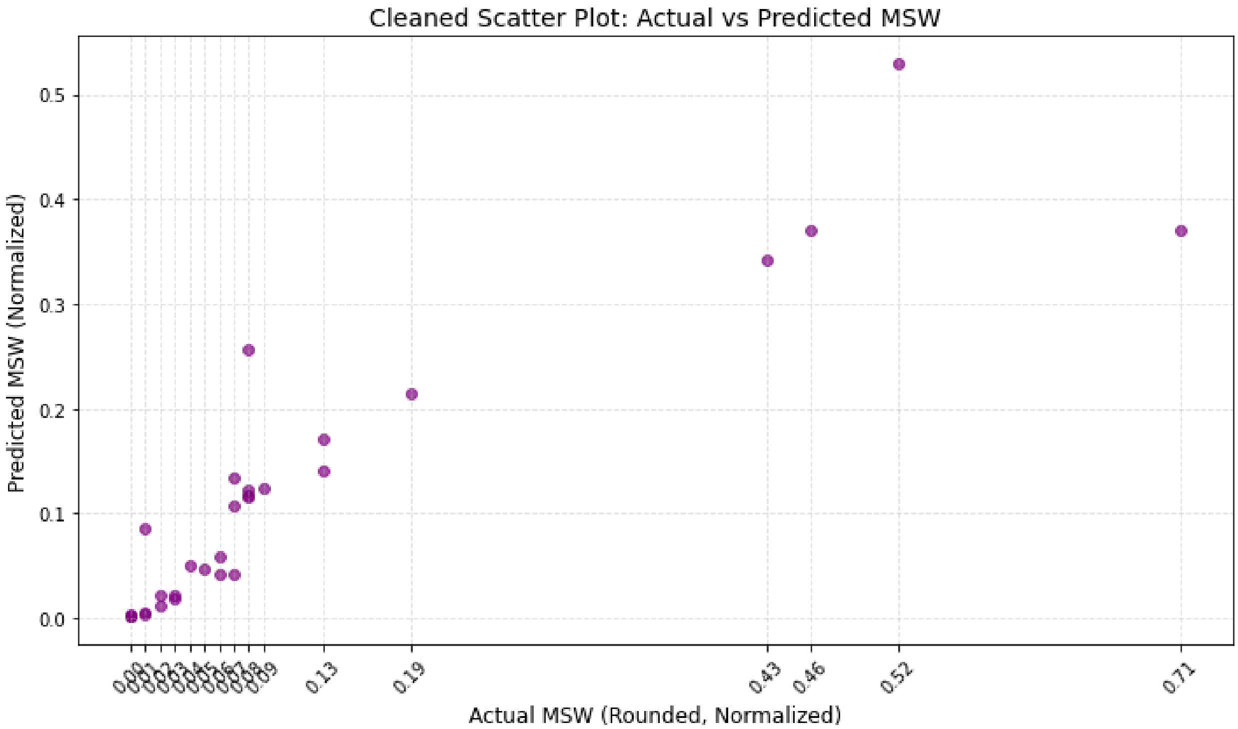

```
# Step 7: Evaluate
mse = mean_squared_error(y_test, y2_pred)
r2 = r2_score(y_test, y2_pred)
print(f"Mean Squared Error: {mse:.6f}")
print(f"R-squared: {r2:.6f}")

# Step 8: Graph – Cleaned Scatter Plot (95% threshold)
threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y2_pred[filtered_indices]

# Sort for smoother plotting
```

```
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x-axis values to reduce clutter
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Unique x values for ticks
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()

Mean Squared Error: 0.009896
R-squared: 0.855082
```



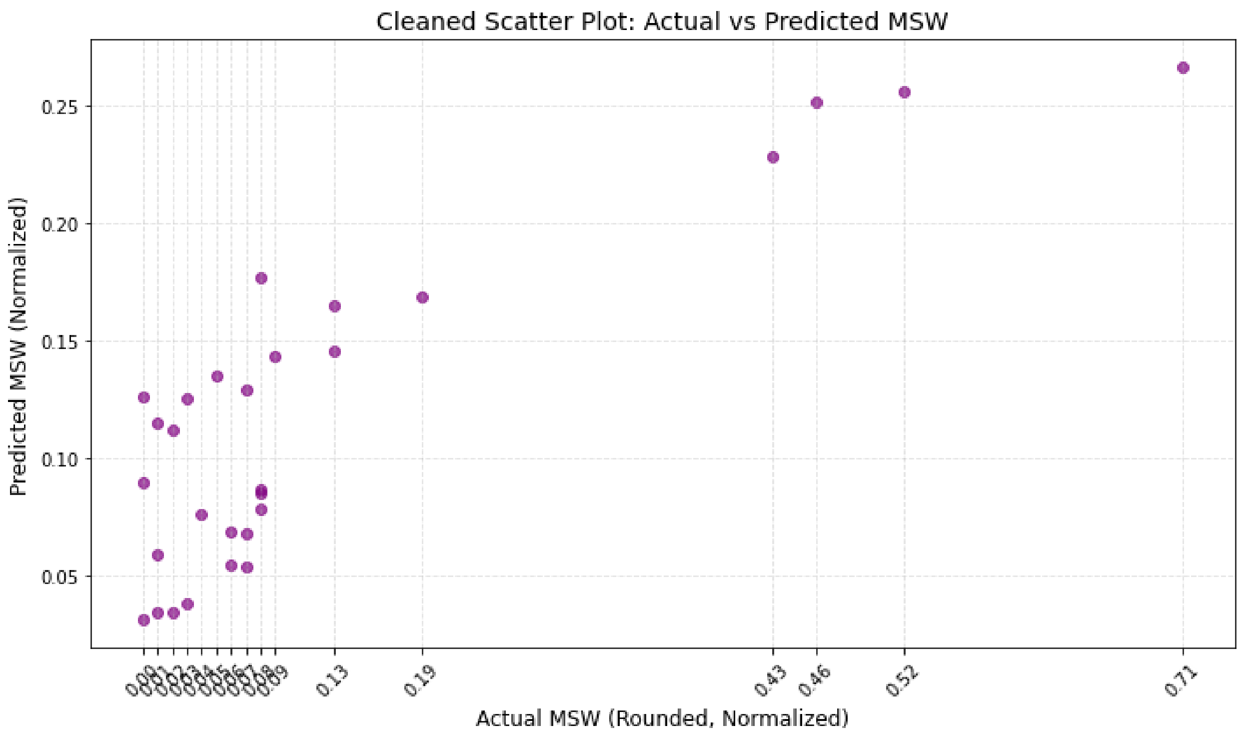
```python
# Step 7: Evaluate
mse = mean_squared_error(y_test, y3_pred)
r2 = r2_score(y_test, y3_pred)
print(f"Mean Squared Error: {mse:.6f}")
print(f"R-squared: {r2:.6f}")

# Step 8: Graph – Cleaned Scatter Plot (95% threshold)
threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y3_pred[filtered_indices]

# Sort for smoother plotting
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x-axis values to reduce clutter
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
fontsize=14)

# Unique x values for ticks
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

```
Mean Squared Error: 0.008669
R-squared: 0.873040
```




```python
# Step 7: Evaluate
mse = mean_squared_error(y_test, y4_pred)
r2 = r2_score(y_test, y4_pred)
print(f"Mean Squared Error: {mse:.6f}")
print(f"R-squared: {r2:.6f}")

# Step 8: Graph – Cleaned Scatter Plot (95% threshold)
threshold = y_test.quantile(0.95)
filtered_indices = y_test <= threshold
y_test_filtered = y_test[filtered_indices]
y_pred_filtered = y4_pred[filtered_indices]

# Sort for smoother plotting
sorted_indices = y_test_filtered.argsort()
y_test_sorted = y_test_filtered.iloc[sorted_indices]
y_pred_sorted = y_pred_filtered[sorted_indices]

# Round x-axis values to reduce clutter
x_vals = np.round(y_test_sorted.values, 2)

plt.figure(figsize=(10, 6))
plt.scatter(x_vals, y_pred_sorted, color='purple', alpha=0.7)

# Labels and title
plt.xlabel('Actual MSW (Rounded, Normalized)', fontsize=12)
plt.ylabel('Predicted MSW (Normalized)', fontsize=12)
plt.title('Cleaned Scatter Plot: Actual vs Predicted MSW',
```

```
fontsize=14)

# Unique x values for ticks
unique_x = sorted(set(x_vals))
plt.xticks(unique_x, rotation=45)

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()

Mean Squared Error: 0.034843
R-squared: 0.489747
```




```
#So the best model came out to be the xgbregressor
```