Part1: Cleaning, wrangling data & Exploratory Data Analysis (EDA)

Data cleaning focuses on removing inaccurate data from your data set whereas data wrangling focuses on transforming the data's format, typically by converting “raw” data into another format more suitable for use.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Here are the steps for EDA:

Steps in EDA:

1. Provide descriptions of your sample and features
2. Check for missing data
3. Identify the shape of your data
4. Identify significant correlations
5. Spot/deal with outliers in the dataset

1.1 Provide descriptions of your sample and features

In [1]:
```
#Read in libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn import preprocessing
import pandas as pd
```

In [2]:
```
#Read the csv file
import pandas as pd

# Specify the file path
file_path = "WaterQltySys.csv"

# Read the CSV file into a pandas DataFrame
try:
    df = pd.read_csv(file_path)
    print("File read successfully.")
    print(df.head())  # Display the first few rows of the DataFrame
except FileNotFoundError:
    print(f"File '{file_path}' not found.")
except Exception as e:
    print("An error occurred:", e)
```

```
File read successfully.
                created_at  entry_id  Tempareture  TDS  Turbidity  pH
0  2024-05-01 15:28:55 UTC       347        20.62  0.0       2.45   0
1  2024-05-01 15:29:11 UTC       348        20.56  0.0       2.47   0
2  2024-05-01 15:29:27 UTC       349        20.62  0.0       2.50   0
3  2024-05-01 15:29:43 UTC       350        20.69  0.0       2.48   0
4  2024-05-01 15:30:00 UTC       351        20.69  0.0       2.48   0
```

In [3]:
```
#Counting the number of rows and columns
rows = len(df.axes[0])
cols = len(df.axes[1])
print("Number of Rows: " + str(rows))
print("Number of Columns: " + str(cols))
```

```
Number of Rows: 100
Number of Columns: 6
```

In [4]:
```
#Representing the datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   created_at   100 non-null    object 
 1   entry_id     100 non-null    int64  
 2   Tempareture  100 non-null    float64
 3   TDS          100 non-null    float64
 4   Turbidity    100 non-null    float64
 5   pH           100 non-null    int64  
dtypes: float64(3), int64(2), object(1)
memory usage: 4.8+ KB
```

In [5]:
```
#Drop null values
df1=df.dropna(axis=1)
```

In [6]:
```
df1.info()
df1.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   created_at   100 non-null    object 
 1   entry_id     100 non-null    int64  
 2   Tempareture  100 non-null    float64
 3   TDS          100 non-null    float64
 4   Turbidity    100 non-null    float64
 5   pH           100 non-null    int64  
dtypes: float64(3), int64(2), object(1)
memory usage: 4.8+ KB
```

Out[6]:

	created_at	entry_id	Tempareture	TDS	Turbidity	pH
0	2024-05-01 15:28:55 UTC	347	20.62	0.0	2.45	0
1	2024-05-01 15:29:11 UTC	348	20.56	0.0	2.47	0
2	2024-05-01 15:29:27 UTC	349	20.62	0.0	2.50	0
3	2024-05-01 15:29:43 UTC	350	20.69	0.0	2.48	0
4	2024-05-01 15:30:00 UTC	351	20.69	0.0	2.48	0

1.2 Check for missing data

In [7]:
```
#check for missing data
df.isnull().sum()
```

Out[7]:
```
created_at     0
entry_id       0
Tempareture    0
TDS            0
Turbidity      0
pH             0
dtype: int64
```

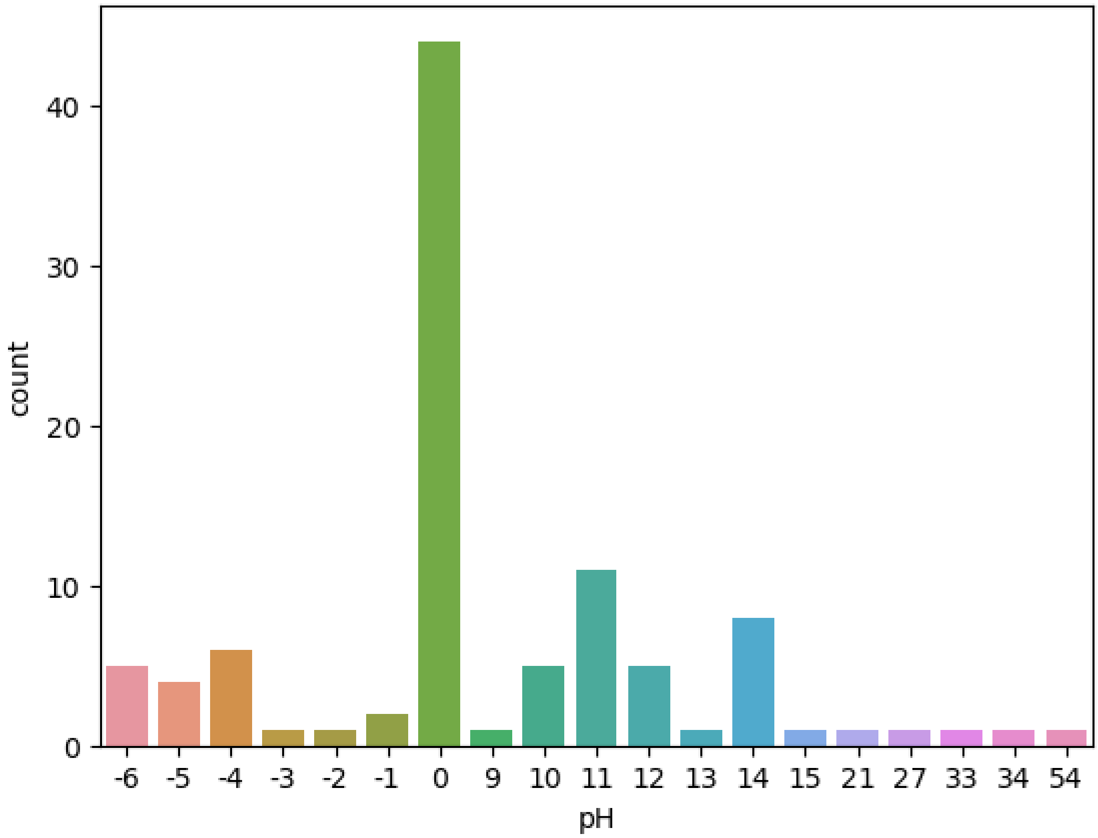
In [8]:
```
#checking for duplicate values
df.duplicated().sum()
```

Out[8]: 0

1.3 Identify the shape of your data

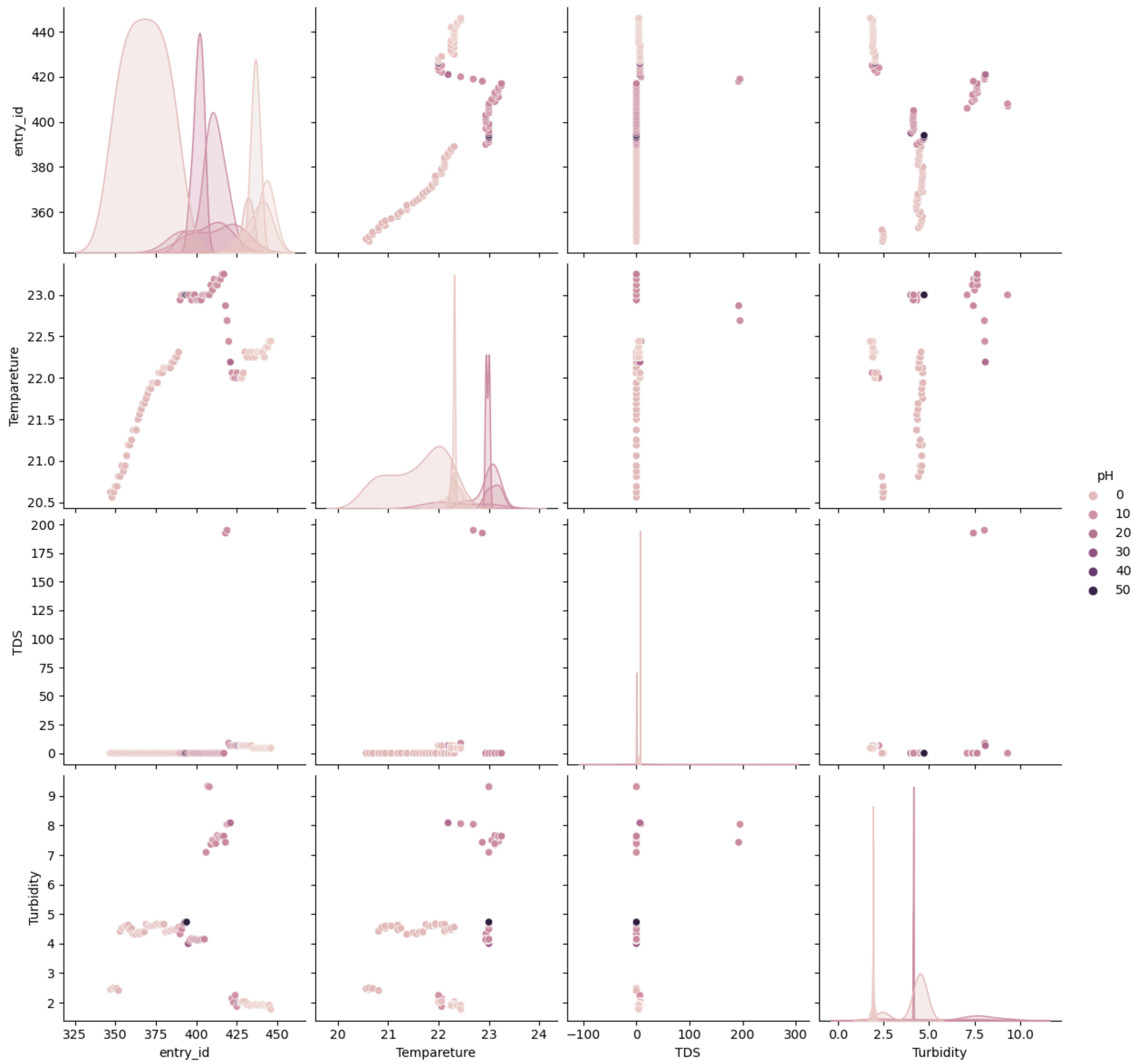
In [9]:
```
#Identifying shape of data
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#countplot
sns.countplot(x='pH', data=df, )
```

Out[9]: <Axes: xlabel='pH', ylabel='count'>



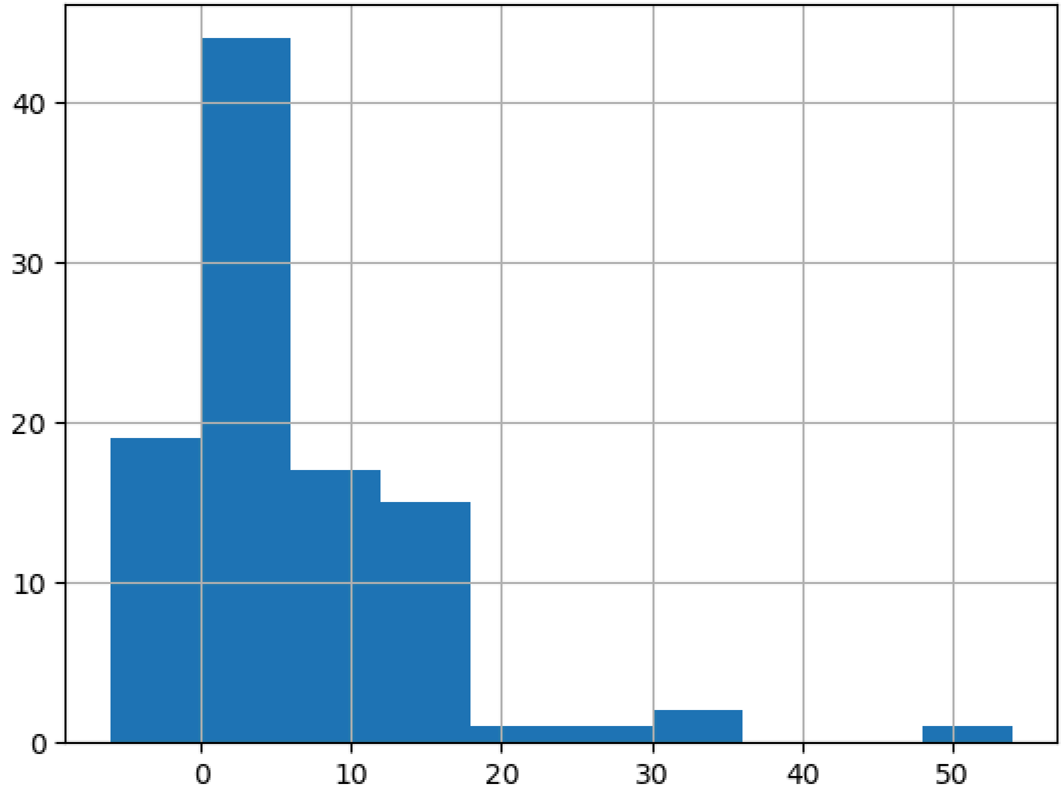
In [10]:
```
#pairplot
sns.pairplot(df, hue="pH",height=3)
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x159314790>



In [11]:
```
#histogram
df.pH.hist()
```

Out[11]: <Axes: >

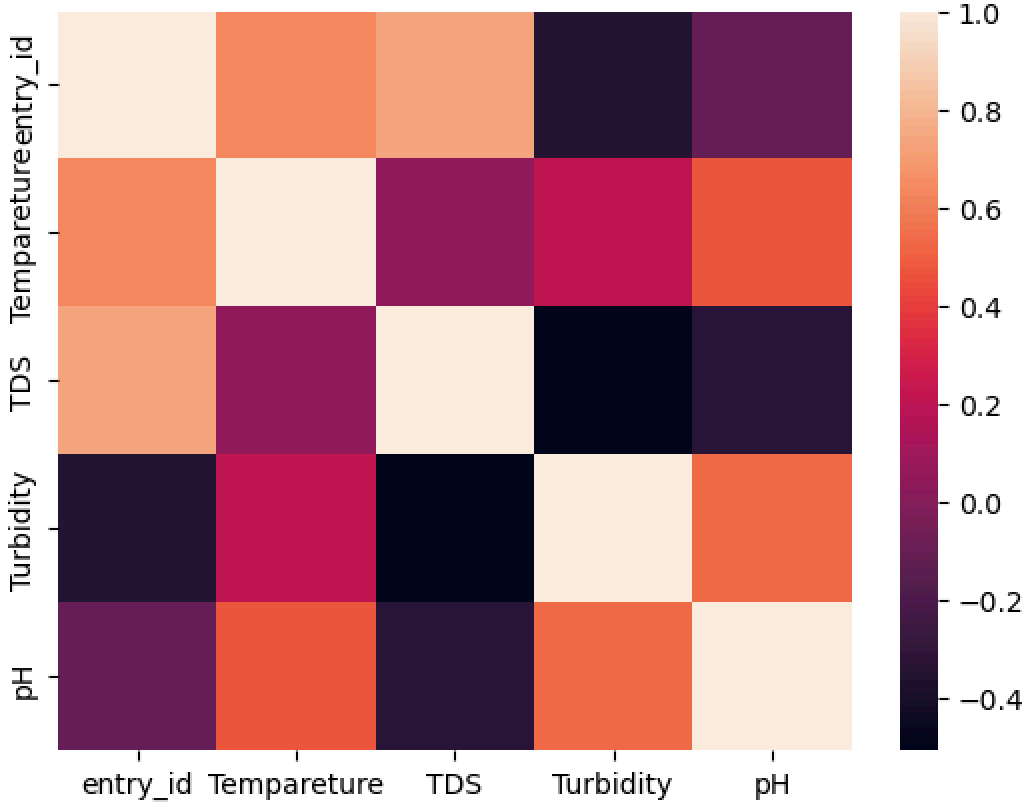


1.4 Identify significant correlations

In [12]:
```
corr = df.corr(method='spearman')
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

```
/var/folders/dp/f1w59vsn4vqbvdmprd4rcvjw0000gn/T/ipykernel_8434/2971803
568.py:1: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Se
lect only valid columns or specify the value of numeric_only to silence
this warning.
  corr = df.corr(method='spearman')
```

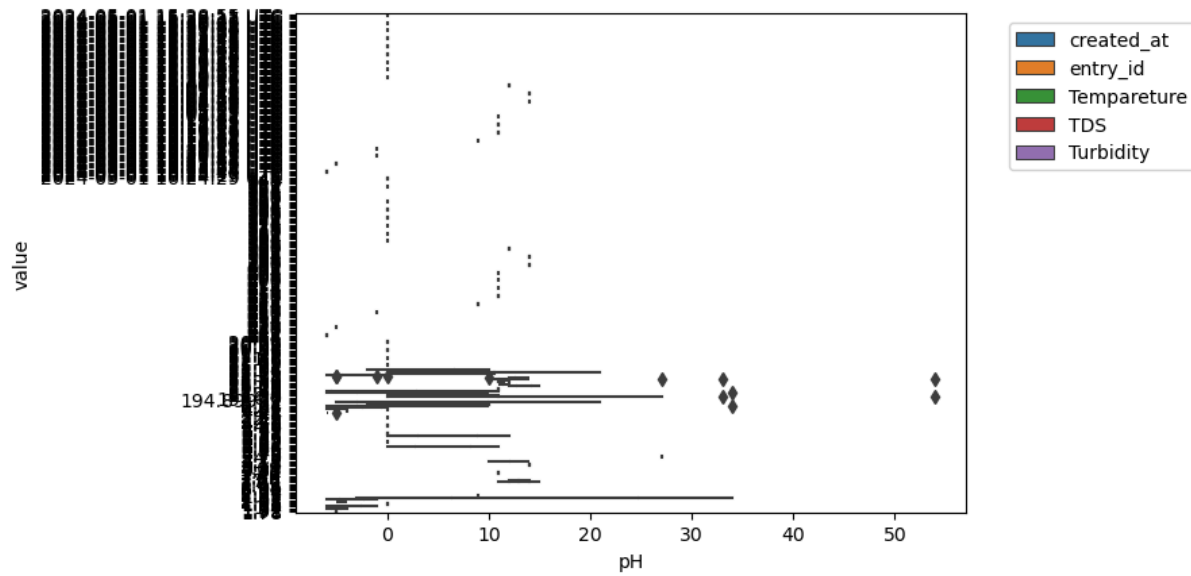
Out[12]: <Axes: >



1.5 Spot/deal with outliers in the dataset

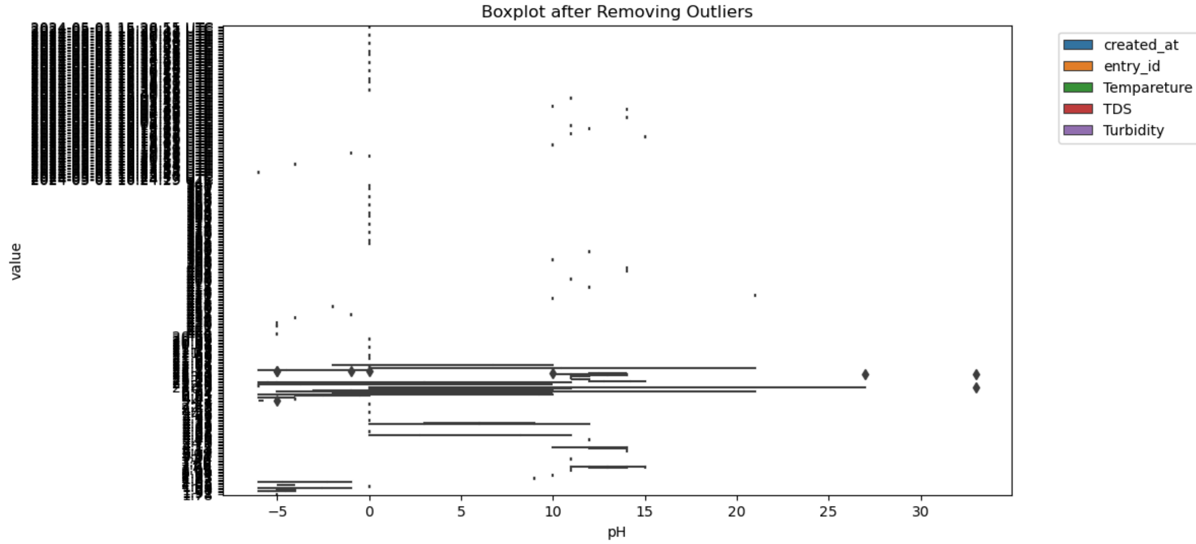
In [13]:
```
#Detecting outliers
dfm = pd.melt(df, id_vars=["pH"])
sns.boxplot(data=dfm, x="pH", y="value", hue="variable", dodge=True)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2)
```

Out[13]: <matplotlib.legend.Legend at 0x105ba1090>



In [14]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Specify the file path
file_path = "WaterQltySys.csv"

# Read the CSV file into a pandas DataFrame
try:
    df = pd.read_csv(file_path)
    print("File read successfully.")

    # Drop rows with missing values
    df = df.dropna()

    print("Rows with missing values dropped.")


    # Removing outliers using z-score
    numeric_cols = df.select_dtypes(include='number').columns
    z_scores = stats.zscore(df[numeric_cols])
    abs_z_scores = abs(z_scores)
    filtered_entries = (abs_z_scores < 3).all(axis=1)
    df = df[filtered_entries]

    print("Outliers removed.")

    # Displaying boxplot after removing outliers
    dfm = pd.melt(df, id_vars=["pH"])
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=dfm, x="pH", y="value", hue="variable", dodge=True)
    plt.title("Boxplot after Removing Outliers")
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2)
    plt.show()

    print(df.head())  # Display the DataFrame after removing outliers

except FileNotFoundError:
    print(f"File '{file_path}' not found.")
except Exception as e:
    print("An error occurred:", e)
```

```
File read successfully.
Rows with missing values dropped.
Outliers removed.
```




```
               created_at  entry_id  Tempareture  TDS  Turbidity  pH
0  2024-05-01 15:28:55 UTC       347        20.62  0.0       2.45   0
1  2024-05-01 15:29:11 UTC       348        20.56  0.0       2.47   0
2  2024-05-01 15:29:27 UTC       349        20.62  0.0       2.50   0
3  2024-05-01 15:29:43 UTC       350        20.69  0.0       2.48   0
4  2024-05-01 15:30:00 UTC       351        20.69  0.0       2.48   0
```

Part 2: Multiple Regression Analysis

Multiple regression analysis is a statistical technique used to understand the relationship between a dependent variable and two or more independent variables.

2.1 Multiple Regression

In [15]:
```
from sklearn import linear_model
import statsmodels.api as sm
import pandas as pd
```

In [16]:
```
# Add a constant term to the independent variables (intercept)
df = sm.add_constant(df)
```

In [17]:
```
# Define the dependent variable and the independent variables
Y=df['pH']
X=df[['entry_id', 'Tempareture','TDS','Turbidity', 'created_at']]
X=df.drop(columns='pH')
X
```

Out[17]:

	const	created_at	entry_id	Tempareture	TDS	Turbidity
0	1.0	2024-05-01 15:28:55 UTC	347	20.62	0.00	2.45
1	1.0	2024-05-01 15:29:11 UTC	348	20.56	0.00	2.47
2	1.0	2024-05-01 15:29:27 UTC	349	20.62	0.00	2.50
3	1.0	2024-05-01 15:29:43 UTC	350	20.69	0.00	2.48
4	1.0	2024-05-01 15:30:00 UTC	351	20.69	0.00	2.48
...	...	...	...	...	...	...
95	1.0	2024-05-01 16:23:24 UTC	442	22.25	4.41	1.90
96	1.0	2024-05-01 16:23:40 UTC	443	22.37	4.40	1.91
97	1.0	2024-05-01 16:23:57 UTC	444	22.37	4.40	1.91
98	1.0	2024-05-01 16:24:13 UTC	445	22.44	4.40	1.92
99	1.0	2024-05-01 16:24:29 UTC	446	22.44	4.40	1.78

96 rows × 6 columns

In [18]:
```
import pandas as pd

# Assuming 'pH' is the dependent variable and 'Temperature', 'TDS', and
# Check data types
print("Data Types:")
print("pH:", df['pH'].dtype)
print("Tempareture:", df['Tempareture'].dtype)
print("TDS:", df['TDS'].dtype)
print("Turbidity:", df['Turbidity'].dtype)
```

```
Data Types:
pH: int64
Tempareture: float64
TDS: float64
Turbidity: float64
```

In [19]:
```
import statsmodels.api as sm

# Assuming 'df' is your DataFrame containing the data

# Define the dependent variable (target)
Y = df['pH']

# Define the independent variables (features)
X = df[['Tempareture', 'TDS', 'Turbidity']]

# Add a constant term to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the OLS model
model = sm.OLS(Y, X).fit()

# Print the summary of the model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:                     pH   R-squared:
0.463
Model:                            OLS   Adj. R-squared:
0.446
Method:                 Least Squares   F-statistic:
26.47
Date:                Tue, 07 May 2024   Prob (F-statistic):
1.98e-12
Time:                        23:02:42   Log-Likelihood:
-301.82
No. Observations:                  96   AIC:
611.6
Df Residuals:                      92   BIC:
621.9
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
========
                  coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
const        -111.3464     19.593     -5.683      0.000    -150.259
-72.434
Tempareture     4.9766      0.922      5.395      0.000       3.145
6.809
TDS            -0.3203      0.272     -1.180      0.241      -0.860
0.219
Turbidity       1.2530      0.411      3.051      0.003       0.437
2.069
==============================================================================
=======
Omnibus:                       38.854   Durbin-Watson:
0.415
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
81.329
Skew:                           1.580   Prob(JB):
2.19e-18
Kurtosis:                       6.217   Cond. No.
759.
==============================================================================
=======

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

In [20]:

```
# Filter significant variables from model1 based on a 90% confidence leve
significant_variables = model.pvalues[model.pvalues <= 0.1].index
```

In [21]:
```
model2 = df[significant_variables]
model2['pH'] = df['pH']
Y_model2 = model2['pH']
X_model2 = model2.drop(columns=['pH'])
model2 = sm.OLS(Y_model2, X_model2).fit()
print(model2.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:                     pH   R-squared:
0.455
Model:                            OLS   Adj. R-squared:
0.443
Method:                 Least Squares   F-statistic:
38.84
Date:                Tue, 07 May 2024   Prob (F-statistic):
5.47e-13
Time:                        23:02:42   Log-Likelihood:
-302.54
No. Observations:                  96   AIC:
611.1
Df Residuals:                      93   BIC:
618.8
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
========
                  coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
const        -105.7134     19.042     -5.552      0.000    -143.527
-67.900
Tempareture     4.6487      0.881      5.274      0.000       2.898
6.399
Turbidity       1.5257      0.340      4.485      0.000       0.850
2.201
==============================================================================
=======
Omnibus:                       38.567   Durbin-Watson:
0.416
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
82.482
Skew:                           1.550   Prob(JB):
1.23e-18
Kurtosis:                       6.318   Cond. No.
735.
==============================================================================
=======

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

```
/var/folders/dp/f1w59vsn4vqbvdmprd4rcvjw0000gn/T/ipykernel_8434/3191613
403.py:2: 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 (ht
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy)
  model2['pH'] = df['pH']
```

2.2 Compare the two models with ANOVA

In [22]:

```
#Comparing two models by ANOVA
import statsmodels.api as sm
from statsmodels.stats.anova import anova_lm
```

In [23]:

```
# Fit Model1 and Model2 as described in previous responses
model = sm.OLS(Y, X).fit()
model2 = sm.OLS(Y_model2, X_model2).fit()
```

In [24]:

```
# Perform ANOVA to compare the two models
anova_results = anova_lm(model, model2)
```

In [25]:

```
# Print the ANOVA table
print(anova_results)
```

```
   df_resid          ssr  df_diff    ss_diff         F  Pr(>F)
0      92.0  3024.050352      0.0        NaN       NaN     NaN
1      93.0  3069.796199     -1.0 -45.745846  1.385878     NaN
```

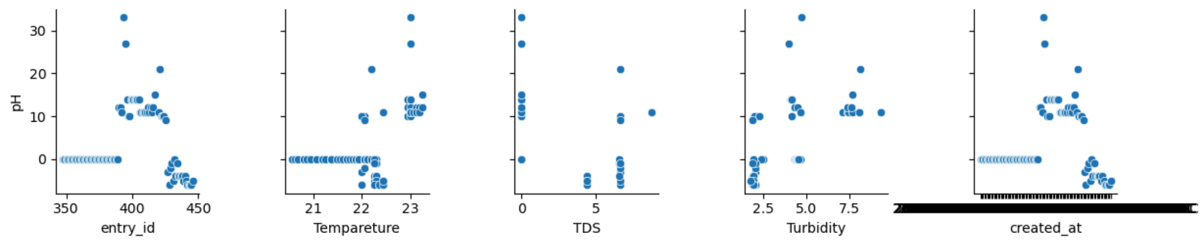
Comparing these metrics, Model 1 has a slightly higher R-squared and F-statistic, indicating that it explains a slightly greater proportion of the variance in the dependent variable (pH) and is a better fit overall. Additionally, Model 1 includes an extra variable (TDS), which might be valuable in explaining pH variability.

Therefore, based on these metrics, Model 1 appears to be the better choice.

2.3 Checking assumptions

In [26]:
```
#Checking Assumptions
import seaborn as sns

# Create a pair plot to visualize relationships
sns.pairplot(df, x_vars=['entry_id', 'Tempareture','TDS','Turbidity', 'c
```

Out[26]: <seaborn.axisgrid.PairGrid at 0x15e7f34d0>

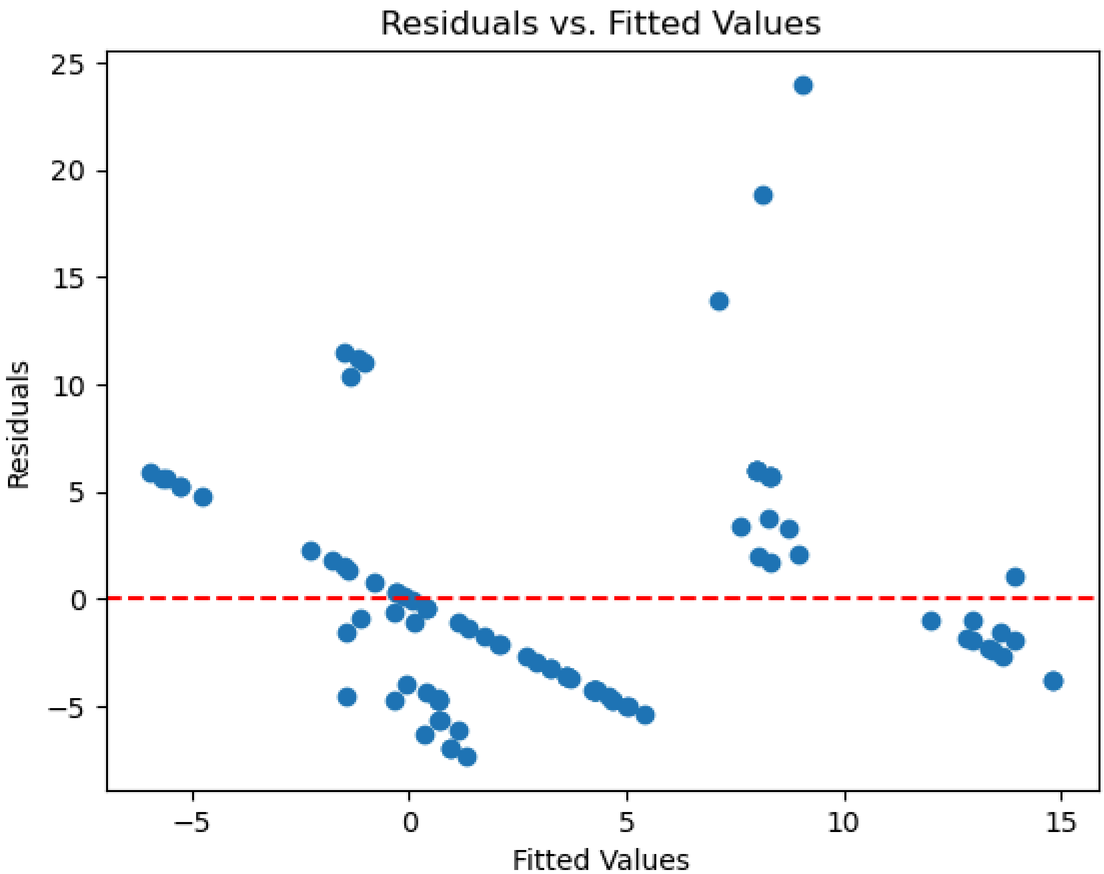


In [27]:
```
#Independence of the errors
from statsmodels.stats.stattools import durbin_watson

# Calculate the Durbin-Watson statistic
dw_statistic = durbin_watson(model.resid)
print(dw_statistic)
```

```
0.41514574764171763
```

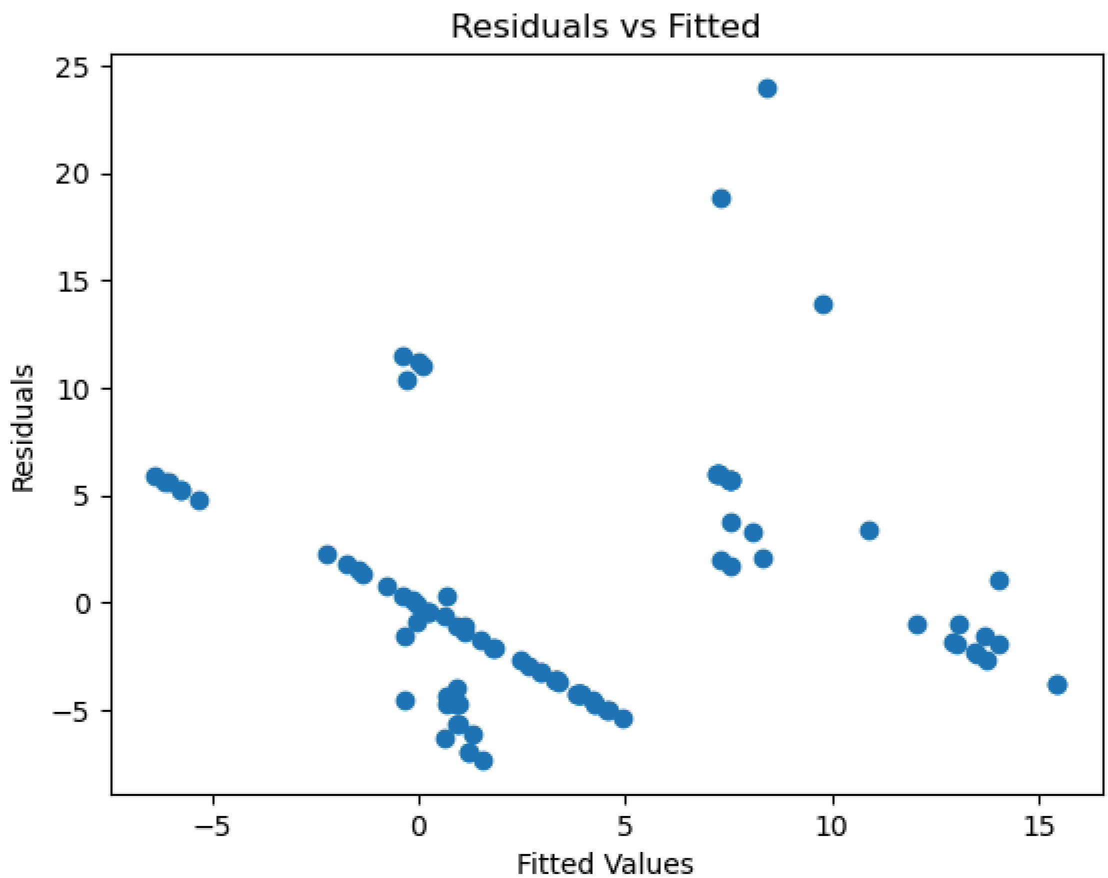
2.4 Homoscedasticity

In [28]:
```
#Homoscedasticity
import statsmodels.api as sm
import statsmodels.stats.api as sms
import matplotlib.pyplot as plt

# Fit your multiple regression model using OLS
model = sm.OLS(endog=Y, exog=X).fit()

# Get the residuals
residuals = model.resid

# Perform statistical tests for homoscedasticity
het_test = sms.het_breuschpagan(residuals, model.model.exog)
white_test = sms.het_white(residuals, model.model.exog)

# Plot residuals vs. fitted values
plt.scatter(model.fittedvalues, residuals)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted Values')

# Add a horizontal line at y=0 for reference
plt.axhline(y=0, color='r', linestyle='--')

plt.show()

# Print the test results
print("Breusch-Pagan test p-value:", het_test[1])
print("White test p-value:", white_test[1])
```




```
Breusch-Pagan test p-value: 0.20810768033725904
White test p-value: 0.04223407633145088
```

In [29]:
```
import matplotlib.pyplot as plt

plt.scatter(model2.fittedvalues, residuals)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted")
plt.show()
```

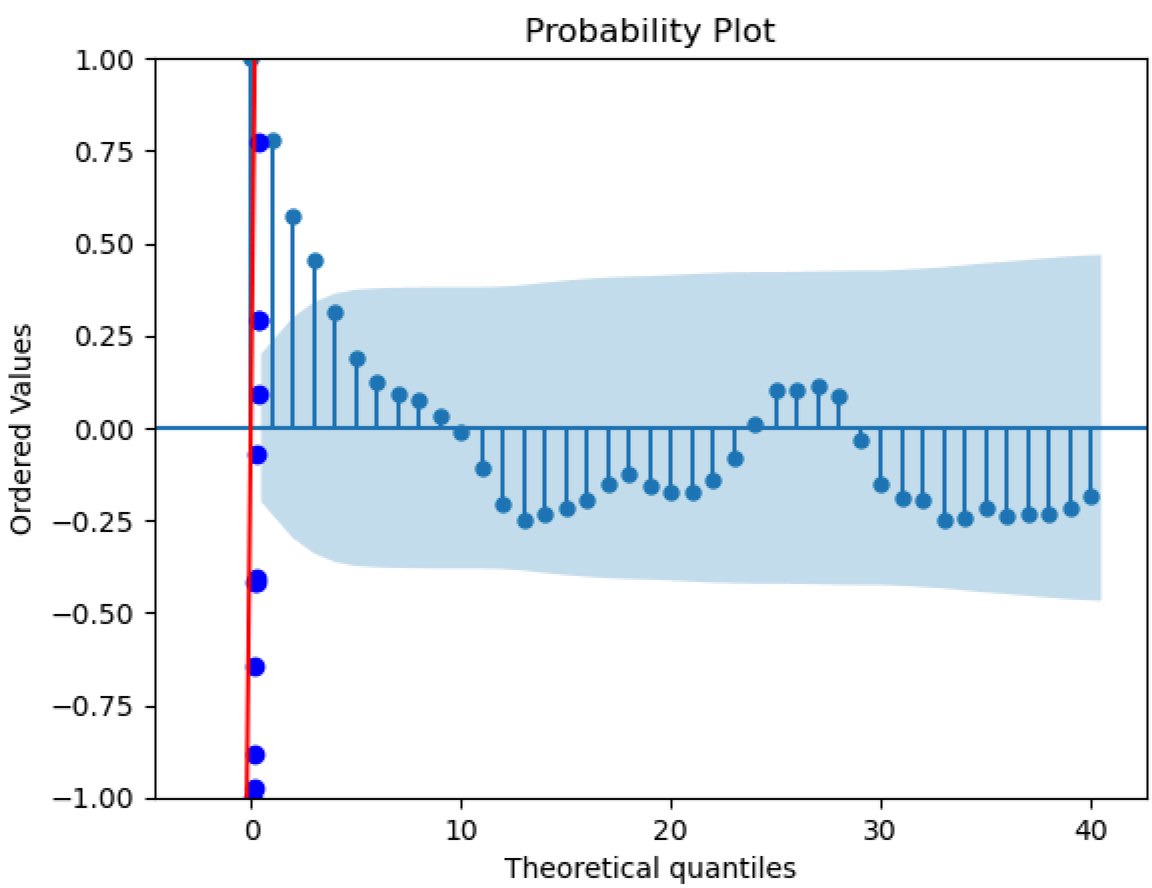


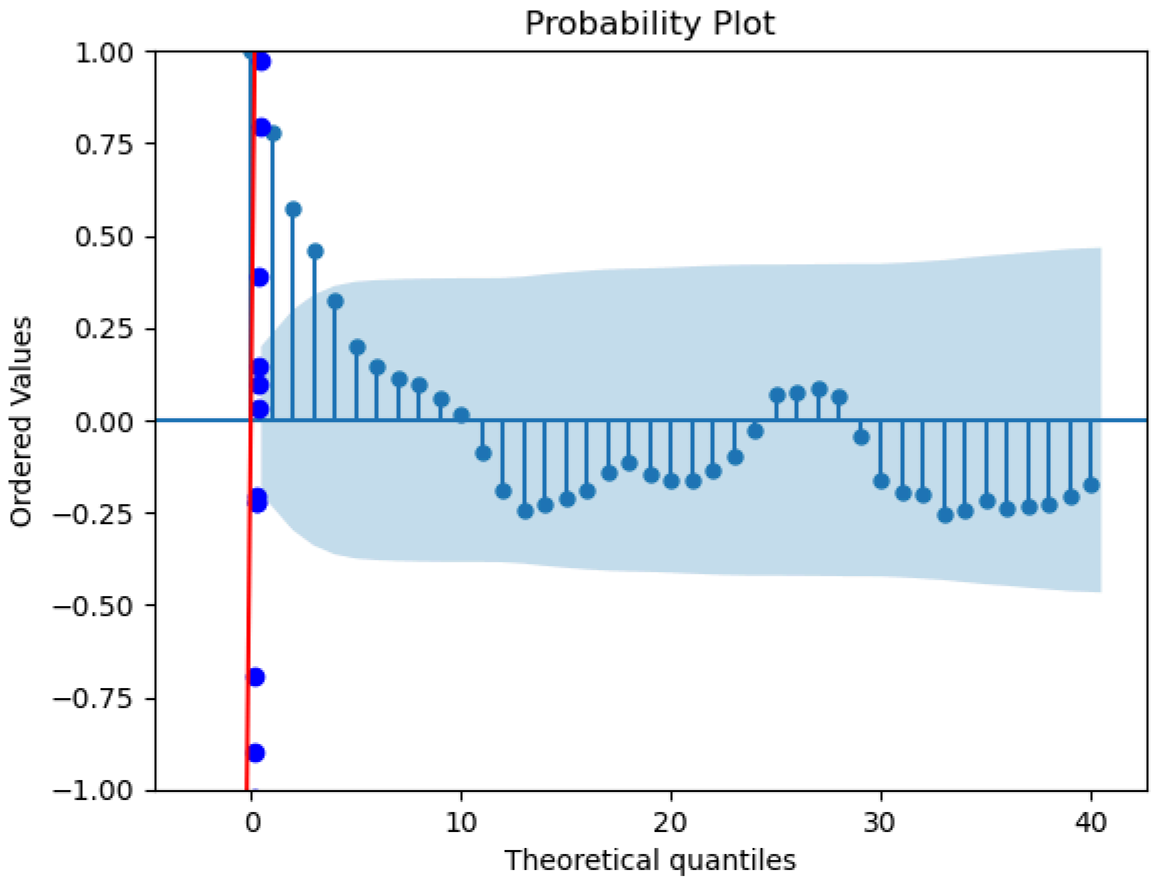
2.5 Normality of Residuals

In [30]:
```
#Normality of Residuals:
import statsmodels.api as sm
import scipy.stats as stats

# Create a histogram of residuals
sm.graphics.tsa.plot_acf(model.resid, lags=40)

# Create a Q-Q plot of residuals
stats.probplot(model.resid, dist="norm", plot=plt)

# Create a histogram of residuals
sm.graphics.tsa.plot_acf(model2.resid, lags=40)

# Create a Q-Q plot of residuals
stats.probplot(model2.resid, dist="norm", plot=plt)
```

Out[30]:
```
((array([-2.44741367, -2.10929279, -1.9135808 , -1.77179758, -1.6586686
1,
        -1.5634916 , -1.48068162, -1.40693772, -1.34013863, -1.2788363
1,
        -1.22199603, -1.16885129, -1.11881757, -1.07143831, -1.0263496
1,
        -0.98325644, -0.94191596, -0.90212572, -0.86371492, -0.8265380
6,
        -0.79046996, -0.75540205, -0.72123945, -0.68789861, -0.6553055
4,
        -0.62339425, -0.59210557, -0.56138615, -0.53118766, -0.5014660
4,
        -0.47218101, -0.44329549, -0.41477525, -0.38658851, -0.3587056
7,
        -0.331099  , -0.30374243, -0.27661134, -0.24968239, -0.2229333
2,
        -0.19634285, -0.1698905 , -0.14355653, -0.11732174, -0.0911674
7,
        -0.06507543, -0.03902762, -0.01300627,  0.01300627,  0.0390276
2,
         0.06507543,  0.09116747,  0.11732174,  0.14355653,  0.1698905
,
         0.19634285,  0.22293332,  0.24968239,  0.27661134,  0.3037424
3,
         0.331099  ,  0.35870567,  0.38658851,  0.41477525,  0.4432954
9,
         0.47218101,  0.50146604,  0.53118766,  0.56138615,  0.5921055
7,
         0.62339425,  0.65530554,  0.68789861,  0.72123945,  0.7554020
5,
         0.79046996,  0.82653806,  0.86371492,  0.90212572,  0.9419159
6,
         0.98325644,  1.02634961,  1.07143831,  1.11881757,  1.1688512
9,
         1.22199603,  1.27883631,  1.34013863,  1.40693772,  1.4806816
2,
         1.5634916 ,  1.65866861,  1.77179758,  1.9135808 ,  2.1092927
9,
         2.44741367]),
  array([-7.532866  , -7.1921997 , -7.1921997 , -6.61909811, -6.3192691
7,
        -5.95904811, -5.94379119, -5.66486886, -5.65526163, -4.9590481
1,
        -4.95636029, -4.94379119, -4.94379119, -4.92853428, -4.8980204
4,
        -4.66486886, -4.61641028, -4.5401257 , -4.44151641, -4.4110025
8,
        -4.26120336, -4.26120336, -4.22567539, -3.93149614, -3.9314961
4,
        -3.91623922, -3.87476631, -3.87476631, -3.84425247, -3.8137386
4,
        -3.4041653 , -3.34313763, -3.32788072, -3.29736688, -2.9719574
9,
        -2.95670057, -2.74635155, -2.67777824, -2.64000471, -2.4903974
,
        -2.47172705, -2.451456  , -2.09636036, -2.01001697, -1.9937484
9,
```

```
         -1.96469779, -1.92842299, -1.83005215, -1.7842814 , -1.6853238
9,
         -1.61909811, -1.48938584, -1.11892201, -1.03951924, -1.0239670
7,
         -0.90102734, -0.69538269, -0.22041103, -0.22041103, -0.2051541
1,
          0.03229531,  0.09938714,  0.14339156,  0.38750223,  0.7934940
4,
          0.97472611,  1.35133871,  1.47339405,  1.75303269,  2.2302949
4,
          2.46156644,  2.68346369,  2.70997494,  3.94283127,  4.4658642
7,
          5.2969352 ,  5.74798145,  5.74798145,  6.04287701,  6.1191615
9,
          6.3675701 ,  6.46156644,  6.46156644,  6.47682336,  6.5225941
1,
          6.75574569,  6.75574569,  6.77100261,  6.80151644,  9.3099267
1,
          9.89798995, 10.00908621, 10.37525221, 11.21579279, 19.6904201
9,
         24.59192219])),
 (5.362364268731166, -1.2252683786321402e-14, 0.9286275999738807))
```



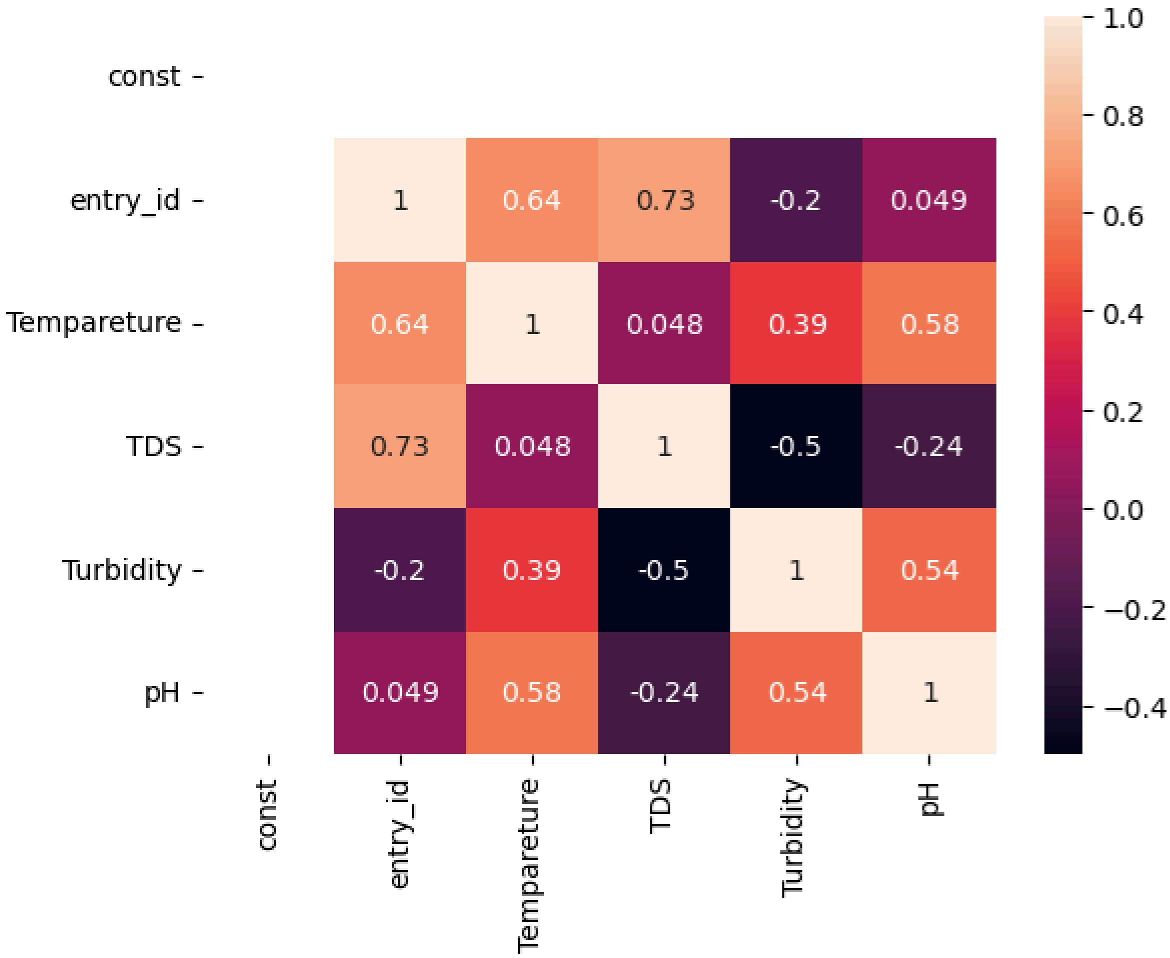


2.6 Multicollinearity

In [31]:
```python
#No or Low Multicollinearity:
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Visualize the correlation matrix
sns.heatmap(correlation_matrix, annot=True)
```

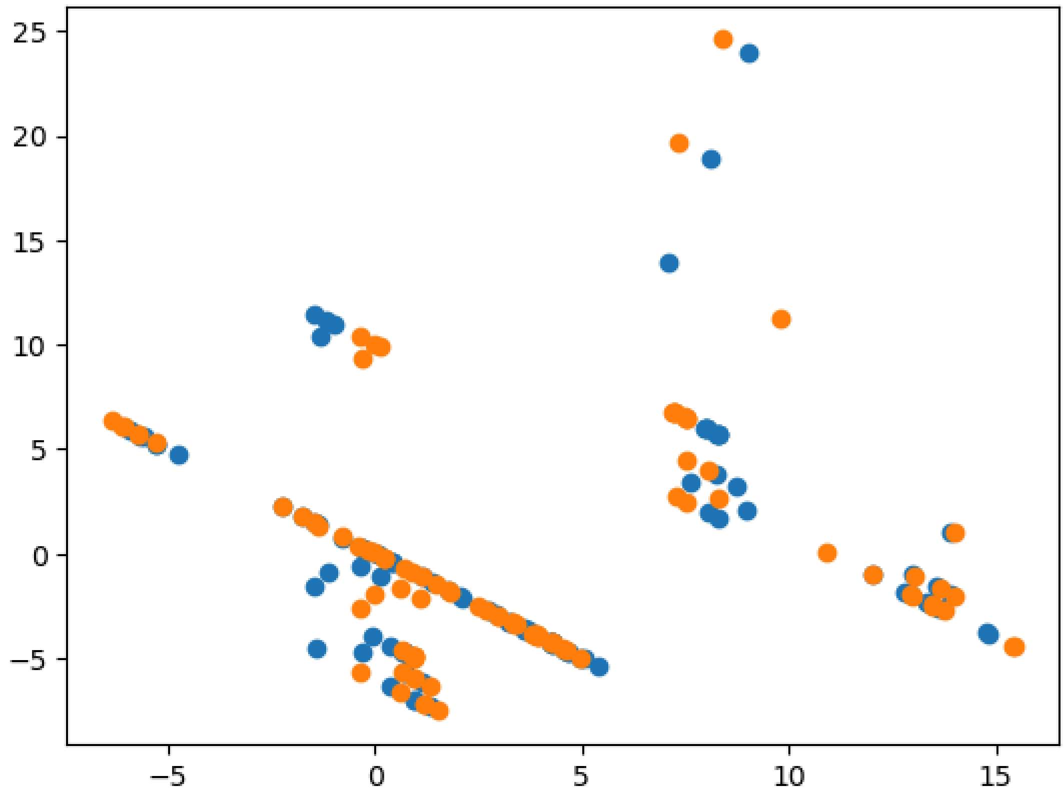
```
/var/folders/dp/f1w59vsn4vqbvdmprd4rcvjw0000gn/T/ipykernel_8434/1388158
39.py:3: FutureWarning: The default value of numeric_only in DataFrame.
corr is deprecated. In a future version, it will default to False. Sele
ct only valid columns or specify the value of numeric_only to silence t
his warning.
  correlation_matrix = df.corr()
```

Out[31]: <Axes: >



In [32]:
```
#No or Low Outliers:
# Create a scatterplot of residuals against predicted values
plt.scatter(model.fittedvalues, model.resid)

#No or Low Outliers:
# Create a scatterplot of residuals against predicted values
plt.scatter(model2.fittedvalues, model2.resid)
```

Out[32]: <matplotlib.collections.PathCollection at 0x15d421310>



Part 3: Feature Selection

There are three types of feature selection techniques. They were:

1. Filter methods
2. Wrapper methods
3. Embedded methods

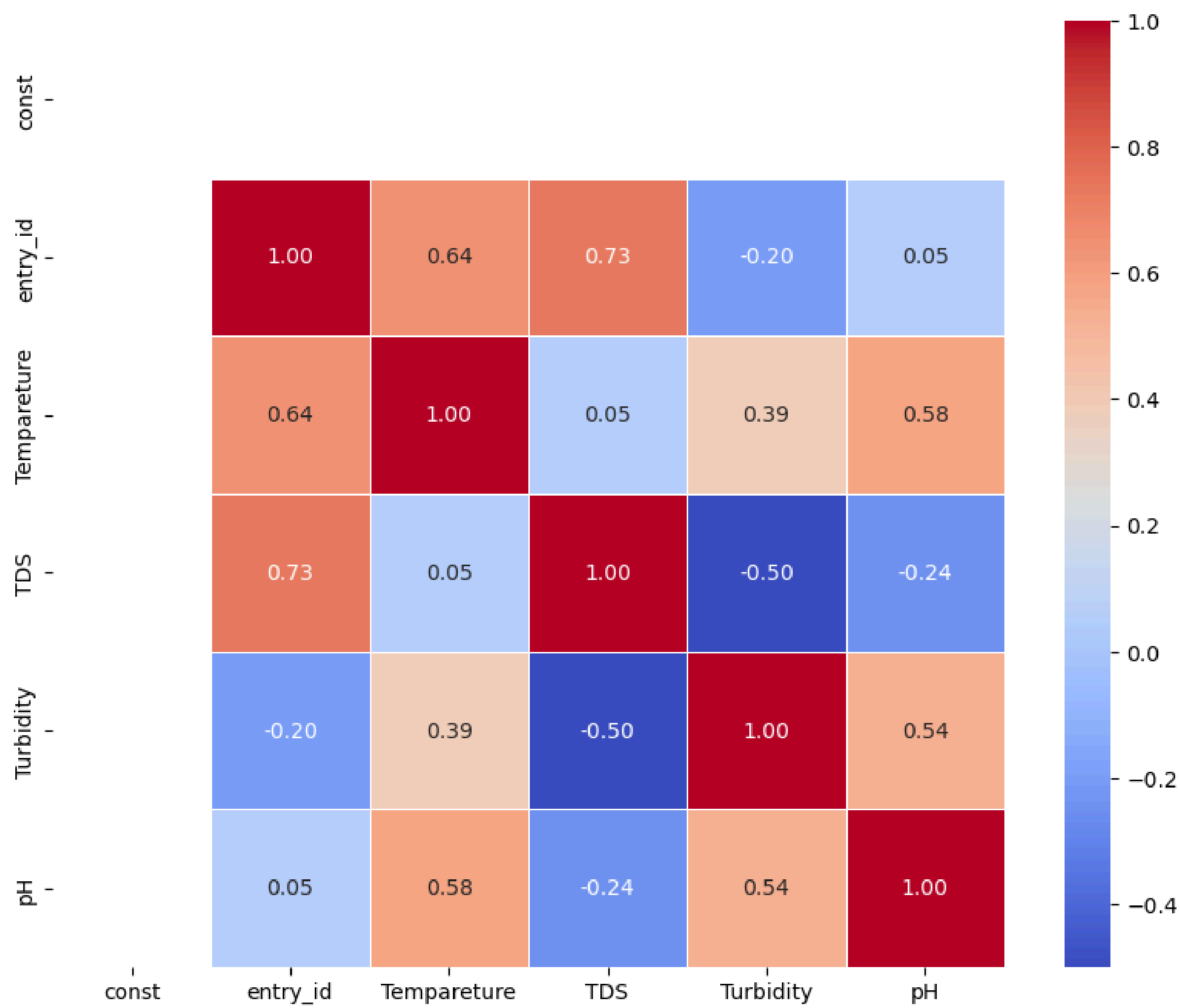
3.1 Filter methods

In [33]:
```
from sklearn.feature_selection import SelectKBest, f_classif
import pandas as pd
import numpy as np


# Instantiate the feature selector
selector = SelectKBest(score_func=f_classif, k='all')

# Fit the selector to your data
fit = selector.fit(X, Y)

# Now we can access the scores and p-values
features_score = pd.DataFrame(fit.scores_)
features_pvalue = pd.DataFrame(np.round(fit.pvalues_, 4))
features = pd.DataFrame(X.columns)
feature_score = pd.concat([features, features_score, features_pvalue], a

# Assigning the column name
feature_score.columns = ["Input_Features", "F_Score", "P_Value"]
print(feature_score.nlargest(30, columns="F_Score"))
```

```
  Input_Features    F_Score  P_Value
3      Turbidity  24.928350      0.0
2            TDS  14.414004      0.0
1    Tempareture  12.300967      0.0
0          const        NaN      NaN

/Users/venkatasrideepthisrikotapeetamabaram/anaconda3/lib/python3.11/si
te-packages/sklearn/feature_selection/_univariate_selection.py:112: Use
rWarning: Features [0] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, Us
erWarning)
/Users/venkatasrideepthisrikotapeetamabaram/anaconda3/lib/python3.11/si
te-packages/sklearn/feature_selection/_univariate_selection.py:113: Run
timeWarning: invalid value encountered in divide
  f = msb / msw
```

In [34]:
```
#Feature Selection using Correlation Matrix with Heatmap (Filtered Method
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Generate a correlation matrix
correlation_matrix = df.corr()

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Create a heatmap with seaborn
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", 

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

```
/var/folders/dp/f1w59vsn4vqbvdmprd4rcvjw0000gn/T/ipykernel_8434/1373878
712.py:7: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Se
lect only valid columns or specify the value of numeric_only to silence
this warning.
  correlation_matrix = df.corr()
```



In [35]:
```
#Feature Slection using Mutual Information(MI) or Information Gain(IG) (
from sklearn.feature_selection import mutual_info_regression
mir = mutual_info_regression(X,Y)
mrs_score = pd.Series(mir,index=X.columns)
mrs_score.sort_values(ascending=False)
```

Out[35]:
```
Turbidity      0.912448
Tempareture    0.823147
TDS            0.482001
const          0.000000
dtype: float64
```

3.2 Wrapper methods

In [36]:
```
#Recurssive Feature Elimination
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate a synthetic dataset for demonstration
X, Y = make_classification(n_samples=1000, n_features=20, random_state=4

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,

# Create a RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Create the RFE model and select 10 features
rfe = RFE(estimator=clf, n_features_to_select=10)
X_train_rfe = rfe.fit_transform(X_train, y_train)
X_test_rfe = rfe.transform(X_test)

# Fit the model on the reduced feature set
clf.fit(X_train_rfe, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test_rfe)

# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with selected features: {accuracy}")

# Print the selected features
selected_features = [f"Feature {i+1}" for i in range(len(rfe.support_)) 
print("Selected Features:", selected_features)
```

```
Accuracy with selected features: 0.91
Selected Features: ['Feature 2', 'Feature 3', 'Feature 6', 'Feature 7',
'Feature 11', 'Feature 12', 'Feature 15', 'Feature 16', 'Feature 17',
'Feature 19']
```

In [37]:
```
#Exhaustive feature selection
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from mlxtend.feature_selection import ExhaustiveFeatureSelector
import numpy as np

# Generate synthetic data for demonstration
np.random.seed(42)
X_synthetic = np.random.rand(100, 5)
y_synthetic = (X_synthetic[:, 0] + X_synthetic[:, 1] > 1).astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_synthetic, y_synthe

# Create a RandomForestClassifier
model = RandomForestClassifier(random_state=42)

# Create ExhaustiveFeatureSelector
efs = ExhaustiveFeatureSelector(model,
                                min_features=1,
                                max_features=X_train.shape[1],
                                scoring='accuracy',
                                print_progress=True,
                                cv=5)

# Fit the selector to the data
efs = efs.fit(X_train, y_train)

# Get the selected feature indices
selected_features = list(efs.best_idx_)

# Use the selected features for training and testing
X_train_selected = X_train[:, selected_features]
X_test_selected = X_test[:, selected_features]

# Train the model with selected features
model.fit(X_train_selected, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_selected)

# Evaluate the model on the test set
accuracy = accuracy_score(y_test, y_pred)
print("Selected features:", selected_features)
print("Accuracy on the test set:", accuracy)
```

```
Features: 31/31

Selected features: [0, 1]
Accuracy on the test set: 0.9
```

In [38]:
```
#Forward Selection
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LogisticRegression as LGR
from sklearn.ensemble import RandomForestClassifier as rfc
```

In [39]:
```
df = pd.DataFrame(columns=['entry_id', 'Tempareture','TDS','Turbidity',
columns = df.columns
feature_names=tuple(df.columns)
feature_names
```

Out[39]:
```
('entry_id', 'Tempareture', 'TDS', 'Turbidity', 'created_at', 'pH')
```

In [40]:
```
X.shape, Y.shape
```

Out[40]:
```
((1000, 20), (1000,))
```

In [41]:
```
sfs1 = SFS(#knn(n_neighbors=3),
           #rfc(n_jobs=8),
           LGR(max_iter=1000),
           k_features='best',
           forward=True,
           floating=False,
           verbose=2,
           #scoring = 'neg_mean_squared_error',  # sklearn regressors
           scoring='accuracy',  # sklearn classifiers
           cv=0)
sfs1 = sfs1.fit(X, Y,feature_names)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 1/20 -- score: 0.867[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  19 out of  19 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 2/20 -- score: 0.877[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 3/20 -- score: 0.881[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  17 out of  17 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 4/20 -- score: 0.883[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 5/20 -- score: 0.884[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 6/20 -- score: 0.884[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 7/20 -- score: 0.886[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 8/20 -- score: 0.886[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 9/20 -- score: 0.885[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
```

```
0.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 10/20 -- score: 0.885[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.1s finished

[2024-05-07 23:02:50] Features: 11/20 -- score: 0.883[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 12/20 -- score: 0.884[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 13/20 -- score: 0.883[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 14/20 -- score: 0.883[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 15/20 -- score: 0.883[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 16/20 -- score: 0.882[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 17/20 -- score: 0.878[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 18/20 -- score: 0.878[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s finished
```

```
[2024-05-07 23:02:50] Features: 19/20 -- score: 0.878[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished

[2024-05-07 23:02:50] Features: 20/20 -- score: 0.877
```

In [42]:
```
sfs1.subsets_
```

Out[42]:
```
{1: {'feature_idx': (5,),
  'cv_scores': array([0.867]),
  'avg_score': 0.867,
  'feature_names': ('5',)},
 2: {'feature_idx': (5, 11),
  'cv_scores': array([0.877]),
  'avg_score': 0.877,
  'feature_names': ('5', '11')},
 3: {'feature_idx': (0, 5, 11),
  'cv_scores': array([0.881]),
  'avg_score': 0.881,
  'feature_names': ('0', '5', '11')},
 4: {'feature_idx': (0, 3, 5, 11),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0', '3', '5', '11')},
 5: {'feature_idx': (0, 3, 5, 10, 11),
  'cv_scores': array([0.884]),
  'avg_score': 0.884,
  'feature_names': ('0', '3', '5', '10', '11')},
 6: {'feature_idx': (0, 3, 5, 7, 10, 11),
  'cv_scores': array([0.884]),
  'avg_score': 0.884,
  'feature_names': ('0', '3', '5', '7', '10', '11')},
 7: {'feature_idx': (0, 3, 5, 7, 10, 11, 19),
  'cv_scores': array([0.886]),
  'avg_score': 0.886,
  'feature_names': ('0', '3', '5', '7', '10', '11', '19')},
 8: {'feature_idx': (0, 3, 5, 7, 8, 10, 11, 19),
  'cv_scores': array([0.886]),
  'avg_score': 0.886,
  'feature_names': ('0', '3', '5', '7', '8', '10', '11', '19')},
 9: {'feature_idx': (0, 3, 5, 7, 8, 9, 10, 11, 19),
  'cv_scores': array([0.885]),
  'avg_score': 0.885,
  'feature_names': ('0', '3', '5', '7', '8', '9', '10', '11', '19')},
 10: {'feature_idx': (0, 3, 5, 7, 8, 9, 10, 11, 12, 19),
  'cv_scores': array([0.885]),
  'avg_score': 0.885,
  'feature_names': ('0', '3', '5', '7', '8', '9', '10', '11', '12', '1
9')},
 11: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '19')},
 12: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 15, 19),
  'cv_scores': array([0.884]),
```

```
  'avg_score': 0.884,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '15',
   '19')},
 13: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '13',
   '15',
   '19')},
 14: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '13',
   '15',
   '16',
   '19')},
 15: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17,
19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
```

```
                 '10',
                 '11',
                 '12',
                 '13',
                 '15',
                 '16',
                 '17',
                 '19')},
               16: {'feature_idx': (0, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 1
              7, 19),
                'cv_scores': array([0.882]),
                'avg_score': 0.882,
                'feature_names': ('0',
                 '3',
                 '4',
                 '5',
                 '6',
                 '7',
                 '8',
                 '9',
                 '10',
                 '11',
                 '12',
                 '13',
                 '15',
                 '16',
                 '17',
                 '19')},
               17: {'feature_idx': (0,
                 1,
                 3,
                 4,
                 5,
                 6,
                 7,
                 8,
                 9,
                 10,
                 11,
                 12,
                 13,
                 15,
                 16,
                 17,
                 19),
                'cv_scores': array([0.878]),
                'avg_score': 0.878,
                'feature_names': ('0',
                 '1',
                 '3',
                 '4',
                 '5',
                 '6',
                 '7',
                 '8',
                 '9',
                 '10',
```

```
     '11',
     '12',
     '13',
     '15',
     '16',
     '17',
     '19')},
   18: {'feature_idx': (0,
     1,
     3,
     4,
     5,
     6,
     7,
     8,
     9,
     10,
     11,
     12,
     13,
     15,
     16,
     17,
     18,
     19),
    'cv_scores': array([0.878]),
    'avg_score': 0.878,
    'feature_names': ('0',
     '1',
     '3',
     '4',
     '5',
     '6',
     '7',
     '8',
     '9',
     '10',
     '11',
     '12',
     '13',
     '15',
     '16',
     '17',
     '18',
     '19')},
   19: {'feature_idx': (0,
     1,
     3,
     4,
     5,
     6,
     7,
     8,
     9,
     10,
     11,
     12,
```

```
     13,
     14,
     15,
     16,
     17,
     18,
     19),
    'cv_scores': array([0.878]),
    'avg_score': 0.878,
    'feature_names': ('0',
     '1',
     '3',
     '4',
     '5',
     '6',
     '7',
     '8',
     '9',
     '10',
     '11',
     '12',
     '13',
     '14',
     '15',
     '16',
     '17',
     '18',
     '19')},
   20: {'feature_idx': (0,
     1,
     2,
     3,
     4,
     5,
     6,
     7,
     8,
     9,
     10,
     11,
     12,
     13,
     14,
     15,
     16,
     17,
     18,
     19),
    'cv_scores': array([0.877]),
    'avg_score': 0.877,
    'feature_names': ('0',
     '1',
     '2',
     '3',
     '4',
     '5',
     '6',
```

```
'7',
'8',
'9',
'10',
'11',
'12',
'13',
'14',
'15',
'16',
'17',
'18',
'19')}}
```

In [43]:
```
sfs1.get_metric_dict()
```

```
/Users/venkatasrideepthisrikotapeetamabaram/anaconda3/lib/python3.11/si
te-packages/numpy/core/_methods.py:269: RuntimeWarning: Degrees of free
dom <= 0 for slice
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/Users/venkatasrideepthisrikotapeetamabaram/anaconda3/lib/python3.11/si
te-packages/numpy/core/_methods.py:261: RuntimeWarning: invalid value e
ncountered in scalar divide
  ret = ret.dtype.type(ret / rcount)
```

Out[43]:
```
{1: {'feature_idx': (5,),
  'cv_scores': array([0.867]),
  'avg_score': 0.867,
  'feature_names': ('5',),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 2: {'feature_idx': (5, 11),
  'cv_scores': array([0.877]),
  'avg_score': 0.877,
  'feature_names': ('5', '11'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 3: {'feature_idx': (0, 5, 11),
  'cv_scores': array([0.881]),
  'avg_score': 0.881,
  'feature_names': ('0', '5', '11'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 4: {'feature_idx': (0, 3, 5, 11),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0', '3', '5', '11'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 5: {'feature_idx': (0, 3, 5, 10, 11),
  'cv_scores': array([0.884]),
  'avg_score': 0.884,
  'feature_names': ('0', '3', '5', '10', '11'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 6: {'feature_idx': (0, 3, 5, 7, 10, 11),
  'cv_scores': array([0.884]),
  'avg_score': 0.884,
  'feature_names': ('0', '3', '5', '7', '10', '11'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 7: {'feature_idx': (0, 3, 5, 7, 10, 11, 19),
  'cv_scores': array([0.886]),
  'avg_score': 0.886,
  'feature_names': ('0', '3', '5', '7', '10', '11', '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 8: {'feature_idx': (0, 3, 5, 7, 8, 10, 11, 19),
  'cv_scores': array([0.886]),
  'avg_score': 0.886,
  'feature_names': ('0', '3', '5', '7', '8', '10', '11', '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 9: {'feature_idx': (0, 3, 5, 7, 8, 9, 10, 11, 19),
```

```
  'cv_scores': array([0.885]),
  'avg_score': 0.885,
  'feature_names': ('0', '3', '5', '7', '8', '9', '10', '11', '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 10: {'feature_idx': (0, 3, 5, 7, 8, 9, 10, 11, 12, 19),
  'cv_scores': array([0.885]),
  'avg_score': 0.885,
  'feature_names': ('0', '3', '5', '7', '8', '9', '10', '11', '12', '1
9'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 11: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0', '3', '4', '5', '7', '8', '9', '10', '11', '1
2', '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 12: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 15, 19),
  'cv_scores': array([0.884]),
  'avg_score': 0.884,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '15',
   '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 13: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '13',
   '15',
   '19'),
  'ci_bound': nan,
```

```
  'std_dev': 0.0,
  'std_err': nan},
 14: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '13',
   '15',
   '16',
   '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 15: {'feature_idx': (0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17,
19),
  'cv_scores': array([0.883]),
  'avg_score': 0.883,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '13',
   '15',
   '16',
   '17',
   '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 16: {'feature_idx': (0, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 1
7, 19),
  'cv_scores': array([0.882]),
  'avg_score': 0.882,
  'feature_names': ('0',
   '3',
   '4',
   '5',
   '6',
   '7',
   '8',
   '9',
   '10',
```

```
   '11',
   '12',
   '13',
   '15',
   '16',
   '17',
   '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 17: {'feature_idx': (0,
   1,
   3,
   4,
   5,
   6,
   7,
   8,
   9,
   10,
   11,
   12,
   13,
   15,
   16,
   17,
   19),
  'cv_scores': array([0.878]),
  'avg_score': 0.878,
  'feature_names': ('0',
   '1',
   '3',
   '4',
   '5',
   '6',
   '7',
   '8',
   '9',
   '10',
   '11',
   '12',
   '13',
   '15',
   '16',
   '17',
   '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan},
 18: {'feature_idx': (0,
   1,
   3,
   4,
   5,
   6,
   7,
   8,
```

```
  9,
  10,
  11,
  12,
  13,
  15,
  16,
  17,
  18,
  19),
 'cv_scores': array([0.878]),
 'avg_score': 0.878,
 'feature_names': ('0',
  '1',
  '3',
  '4',
  '5',
  '6',
  '7',
  '8',
  '9',
  '10',
  '11',
  '12',
  '13',
  '15',
  '16',
  '17',
  '18',
  '19'),
 'ci_bound': nan,
 'std_dev': 0.0,
 'std_err': nan},
19: {'feature_idx': (0,
  1,
  3,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19),
 'cv_scores': array([0.878]),
 'avg_score': 0.878,
 'feature_names': ('0',
  '1',
  '3',
```

```
      '4',
      '5',
      '6',
      '7',
      '8',
      '9',
      '10',
      '11',
      '12',
      '13',
      '14',
      '15',
      '16',
      '17',
      '18',
      '19'),
     'ci_bound': nan,
     'std_dev': 0.0,
     'std_err': nan},
   20: {'feature_idx': (0,
      1,
      2,
      3,
      4,
      5,
      6,
      7,
      8,
      9,
      10,
      11,
      12,
      13,
      14,
      15,
      16,
      17,
      18,
      19),
     'cv_scores': array([0.877]),
     'avg_score': 0.877,
     'feature_names': ('0',
      '1',
      '2',
      '3',
      '4',
      '5',
      '6',
      '7',
      '8',
      '9',
      '10',
      '11',
      '12',
      '13',
      '14',
      '15',
```

```
   '16',
   '17',
   '18',
   '19'),
  'ci_bound': nan,
  'std_dev': 0.0,
  'std_err': nan}}
```

In [44]:
```
sfs1.k_feature_names_, sfs1.k_feature_idx_
```

Out[44]:
```
(('0', '3', '5', '7', '10', '11', '19'), (0, 3, 5, 7, 10, 11, 19))
```

In [45]:
```
df = pd.DataFrame.from_dict(sfs1.get_metric_dict()).T
df[["feature_idx","avg_score"]]
```

Out[45]:

	feature_idx	avg_score
1	(5,)	0.867
2	(5, 11)	0.877
3	(0, 5, 11)	0.881
4	(0, 3, 5, 11)	0.883
5	(0, 3, 5, 10, 11)	0.884
6	(0, 3, 5, 7, 10, 11)	0.884
7	(0, 3, 5, 7, 10, 11, 19)	0.886
8	(0, 3, 5, 7, 8, 10, 11, 19)	0.886
9	(0, 3, 5, 7, 8, 9, 10, 11, 19)	0.885
10	(0, 3, 5, 7, 8, 9, 10, 11, 12, 19)	0.885
11	(0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 19)	0.883
12	(0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 15, 19)	0.884
13	(0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 19)	0.883
14	(0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 19)	0.883
15	(0, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, ...	0.883
16	(0, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 1...	0.882
17	(0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15...	0.878
18	(0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15...	0.878
19	(0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	0.878
20	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	0.877

3.3 Embedded methods

In [46]:
```
#LASSO

from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

# Generate some synthetic data
np.random.seed(42)
X = np.random.rand(100, 5)  # 100 samples, 5 features
Y = 2 * X[:, 0] + 3 * X[:, 1] + 0.5 * X[:, 2] + np.random.randn(100)  # 

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,

# Create a Lasso model
lasso = Lasso(alpha=0.01)

# Fit the model to the training data
lasso.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lasso.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Print the coefficients
print('Coefficients:', lasso.coef_)
```

```
Mean Squared Error: 0.9218687071669031
Coefficients: [ 1.52980748  2.64949441  0.98830069  0.         -0.33410
171]
```

In [47]:

```
#RIDGE regression

from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

# Generate some synthetic data
np.random.seed(42)
X = np.random.rand(100, 5)  # 100 samples, 5 features
Y = 2 * X[:, 0] + 3 * X[:, 1] + 0.5 * X[:, 2] + np.random.randn(100)  # 

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,

# Create a Ridge model
ridge = Ridge(alpha=1.0)

# Fit the model to the training data
ridge.fit(X_train, y_train)

# Make predictions on the test set
y_pred = ridge.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Print the coefficients
print('Coefficients:', ridge.coef_)
```

```
Mean Squared Error: 0.9046958516981591
Coefficients: [ 1.40006201  2.41391459  0.97117803  0.09796465 -0.42814
467]
```

In [48]:
```
#Elastic Net
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

# Generate some synthetic data
np.random.seed(42)
X = np.random.rand(100, 5)  # 100 samples, 5 features
Y = 2 * X[:, 0] + 3 * X[:, 1] + 0.5 * X[:, 2] + np.random.randn(100)  # 

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,

# Create an Elastic Net model
elastic_net = ElasticNet(alpha=1.0, l1_ratio=0.5)

# Fit the model to the training data
elastic_net.fit(X_train, y_train)

# Make predictions on the test set
y_pred = elastic_net.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Print the coefficients
print('Coefficients:', elastic_net.coef_)
```

```
Mean Squared Error: 1.1794384719415933
Coefficients: [ 0.   0.   0. -0. -0.]
```

Implementation of Machine Learning Algorithms

Enhancing Water Quality Monitoring: Integrating Neural Networks for Advanced Analysis

Neural Network Library

In [49]:
```
import numpy as np

class Layer:
    def __init__(self, input_size, output_size):
        self.weights = np.random.randn(input_size, output_size)
        self.biases = np.zeros((1, output_size))
        self.inputs = None
        self.outputs = None

    def forward(self, inputs):
        self.inputs = inputs
        self.outputs = np.dot(inputs, self.weights) + self.biases
        return self.outputs

    def backward(self, gradients):
        # Calculating the gradients w.r.t. weights and biases
        weights_gradients = np.dot(self.inputs.T, gradients)
        biases_gradients = np.sum(gradients, axis=0, keepdims=True)

        # Updating the weights and biases
        self.weights -= learning_rate * weights_gradients
        self.biases -= learning_rate * biases_gradients

        # Calculating the gradients w.r.t. inputs
        return np.dot(gradients, self.weights.T)

class Activation:
    def __init__(self, activation_func, activation_func_derivative):
        self.activation_func = activation_func
        self.activation_func_derivative = activation_func_derivative

    def forward(self, inputs):
        self.inputs = inputs
        return self.activation_func(inputs)

    def backward(self, gradients):
        return gradients * self.activation_func_derivative(self.inputs)

class Loss:
    @staticmethod
    def mean_squared_error(predictions, targets):
        return np.mean((predictions - targets) ** 2)

    @staticmethod
    def mean_squared_error_derivative(predictions, targets):
        return 2 * (predictions - targets) / len(predictions)

class NeuralNetwork:
    def __init__(self):
        self.layers = []

    def add_layer(self, layer):
        self.layers.append(layer)

    def forward(self, inputs):
        output = inputs
        for layer in self.layers:
```

```
            output = layer.forward(output)
        return output

    def train(self, X_train, y_train, learning_rate, epochs):
        for epoch in range(epochs):
            predictions = self.forward(X_train)
            loss = Loss.mean_squared_error(predictions, y_train)
            print(f"Epoch {epoch+1}/{epochs}, Loss: {loss}")

            # Backpropagation
            error = Loss.mean_squared_error_derivative(predictions, y_tra
            for layer in reversed(self.layers):
                error = layer.backward(error)

# Defining the sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

The Layer Class

In [50]:

```
import numpy as np

class DenseLayer(Layer):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.weights = np.random.randn(input_size, output_size)
        self.biases = np.zeros((1, output_size))

    def forward(self, inputs):
        self.inputs = inputs
        self.outputs = np.dot(inputs, self.weights) + self.biases
        return self.outputs

    def backward(self, gradients):
        weights_gradients = np.dot(self.inputs.T, gradients)
        biases_gradients = np.sum(gradients, axis=0, keepdims=True)
        input_gradients = np.dot(gradients, self.weights.T)



        return input_gradients
```

Linear Layer

In [51]:
```
import numpy as np

class Layer:
    def __init__(self):
        self.inputs = None
        self.outputs = None

    def forward(self, inputs):
        pass

    def backward(self, gradients):
        pass

class LinearLayer(Layer):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.weights = np.random.randn(input_size, output_size)
        self.biases = np.zeros((1, output_size))
        self.inputs = None

    def forward(self, inputs):
        self.inputs = inputs
        return np.dot(inputs, self.weights) + self.biases

    def backward(self, gradients):
        weights_gradients = np.dot(self.inputs.T, gradients)
        biases_gradients = np.sum(gradients, axis=0, keepdims=True)
        input_gradients = np.dot(gradients, self.weights.T)



        return input_gradients, weights_gradients, biases_gradients
```

Sigmoid Function

In [52]:
```
import numpy as np

class Layer:
    def __init__(self):
        self.inputs = None
        self.outputs = None

    def forward(self, inputs):
        pass

    def backward(self, gradients):
        pass

class SigmoidLayer(Layer):
    def __init__(self):
        super().__init__()
        self.outputs = None

    def forward(self, inputs):
        self.inputs = inputs
        self.outputs = 1 / (1 + np.exp(-inputs))
        return self.outputs

    def backward(self, gradients):
        sigmoid_derivative = self.outputs * (1 - self.outputs)
        return gradients * sigmoid_derivative
```

Rectified Linear Unit (ReLU)

In [53]:
```
import numpy as np

class Layer:
    def __init__(self):
        self.inputs = None
        self.outputs = None

    def forward(self, inputs):
        pass

    def backward(self, gradients):
        pass

class ReLU(Layer):
    def __init__(self):
        super().__init__()
        self.outputs = None

    def forward(self, inputs):
        self.inputs = inputs
        self.outputs = np.maximum(0, inputs)
        return self.outputs

    def backward(self, gradients):
        relu_derivative = np.where(self.inputs > 0, 1, 0)
        return gradients * relu_derivative
```

Binary Cross-Entropy Loss

In [54]:
```
import numpy as np

class Layer:
    def __init__(self):
        self.inputs = None
        self.outputs = None

    def forward(self, inputs):
        pass

    def backward(self, gradients):
        pass

class BinaryCrossEntropyLoss(Layer):
    def __init__(self):
        super().__init__()

    def forward(self, predictions, targets):
        self.inputs = predictions
        self.targets = targets
        return -np.mean(targets * np.log(predictions + 1e-15) + (1 - tar

    def backward(self):
        return (self.inputs - self.targets) / (self.inputs * (1 - self.i
```

The Sequential Class

In [55]:
```
class Sequential(Layer):
    def __init__(self):
        super().__init__()
        self.layers = []

    def add(self, layer):
        self.layers.append(layer)

    def forward(self, inputs):
        output = inputs
        for layer in self.layers:
            output = layer.forward(output)
        return output

    def backward(self, gradients):
        for layer in reversed(self.layers):
            gradients = layer.backward(gradients)
        return gradients
```

Saving and Loading

In [56]:
```
def save_weights(self, filename):
    weights = [layer.weights for layer in self.layers if hasattr(layer,
    biases = [layer.biases for layer in self.layers if hasattr(layer, 'b
    np.savez(filename, weights=weights, biases=biases)

def load_weights(self, filename):
    data = np.load(filename)
    for layer, weights, biases in zip(self.layers, data['weights'], data
        if hasattr(layer, 'weights'):
            layer.weights = weights
        if hasattr(layer, 'biases'):
            layer.biases = biases
```

Testing the Library

In [57]:
```
import numpy as np

# XOR input data
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

# XOR labels
y = np.array([[0],
              [1],
              [1],
              [0]])

class Layer:
    def __init__(self):
        self.inputs = None
        self.outputs = None

    def forward(self, inputs):
        self.inputs = inputs
        self.outputs = inputs
        return inputs

    def backward(self, gradients):
        pass

class SigmoidLayer(Layer):
    def __init__(self):
        super().__init__()
        self.outputs = None

    def forward(self, inputs):
        self.inputs = inputs
        self.outputs = 1 / (1 + np.exp(-inputs))
        return self.outputs

    def backward(self, gradients):
        sigmoid_derivative = self.outputs * (1 - self.outputs)
        return gradients * sigmoid_derivative

class TanhLayer(Layer):
    def __init__(self):
        super().__init__()
        self.outputs = None

    def forward(self, inputs):
        self.inputs = inputs
        self.outputs = np.tanh(inputs)
        return self.outputs

    def backward(self, gradients):
        tanh_derivative = 1 - np.tanh(self.inputs)**2
        return gradients * tanh_derivative

# Defining the neural network
class XOR_Model:
```

```
    def __init__(self):
        self.hidden_layer = Layer()
        self.output_layer = SigmoidLayer()

    def forward(self, inputs):
        hidden_output = self.hidden_layer.forward(inputs.dot(self.hidden_
        output = self.output_layer.forward(hidden_output.dot(self.output_
        return output

    def backward(self, gradients):
        gradients = self.output_layer.backward(gradients)
        gradients = self.hidden_layer.backward(gradients.dot(self.output_
        return gradients

    def train(self, X, y, learning_rate=0.1, epochs=10000):
        np.random.seed(0)
        self.hidden_weights = np.random.randn(X.shape[1], 2)
        self.hidden_bias = np.zeros((1, 2))
        self.output_weights = np.random.randn(2, 1)
        self.output_bias = np.zeros((1, 1))

        for epoch in range(epochs):
            # Forward pass
            output = self.forward(X)

            # Computing the loss
            loss = np.mean((output - y) ** 2)

            # Backward pass
            gradient = 2 * (output - y) / X.shape[0]
            self.backward(gradient)

            # Updating the weights
            self.hidden_weights -= learning_rate * X.T.dot(self.hidden_la
            self.hidden_bias -= learning_rate * np.sum(self.hidden_layer
            self.output_weights -= learning_rate * self.hidden_layer.out
            self.output_bias -= learning_rate * np.sum(gradient, axis=0,

            if epoch % 1000 == 0:
                print(f"Epoch: {epoch}, Loss: {loss}")

        output[output < 0.5] = 0
        output[output >= 0.5] = 1
        print("\nThresholded Output Matrix:")
        print(output.astype(int))

# Training the model with sigmoid activations
print("Training with sigmoid activations:")
model_sigmoid = XOR_Model()
model_sigmoid.train(X, y)

# Saving the weights
np.savez('XOR_solved_sigmoid.npz', hidden_weights=model_sigmoid.hidden_we
                                    hidden_bias=model_sigmoid.hidden_bias
                                    output_weights=model_sigmoid.output_w
                                    output_bias=model_sigmoid.output_bias
```

```
# Training the model with hyperbolic tangent activations
print("\nTraining with hyperbolic tangent activations:")
class XOR_Model_Tanh(XOR_Model):
    def __init__(self):
        super().__init__()
        self.hidden_layer = TanhLayer()

model_tanh = XOR_Model_Tanh()
model_tanh.train(X, y)

# Saving weights
np.savez('XOR_solved_tanh.npz', hidden_weights=model_tanh.hidden_weights
                                 hidden_bias=model_tanh.hidden_bias,
                                 output_weights=model_tanh.output_weight
                                 output_bias=model_tanh.output_bias)
```

```
Training with sigmoid activations:
Epoch: 0, Loss: 0.3648948783953737
Epoch: 1000, Loss: 0.25
Epoch: 2000, Loss: 0.25
Epoch: 3000, Loss: 0.25
Epoch: 4000, Loss: 0.25
Epoch: 5000, Loss: 0.25
Epoch: 6000, Loss: 0.25
Epoch: 7000, Loss: 0.25
Epoch: 8000, Loss: 0.25
Epoch: 9000, Loss: 0.25

Thresholded Output Matrix:
[[1]
 [1]
 [1]
 [1]]

Training with hyperbolic tangent activations:
Epoch: 0, Loss: 0.2355964084760832
Epoch: 1000, Loss: 0.25
Epoch: 2000, Loss: 0.25
Epoch: 3000, Loss: 0.25
Epoch: 4000, Loss: 0.25
Epoch: 5000, Loss: 0.25
Epoch: 6000, Loss: 0.25
Epoch: 7000, Loss: 0.25
Epoch: 8000, Loss: 0.25
Epoch: 9000, Loss: 0.25

Thresholded Output Matrix:
[[1]
 [1]
 [1]
 [1]]
```

Model Selection

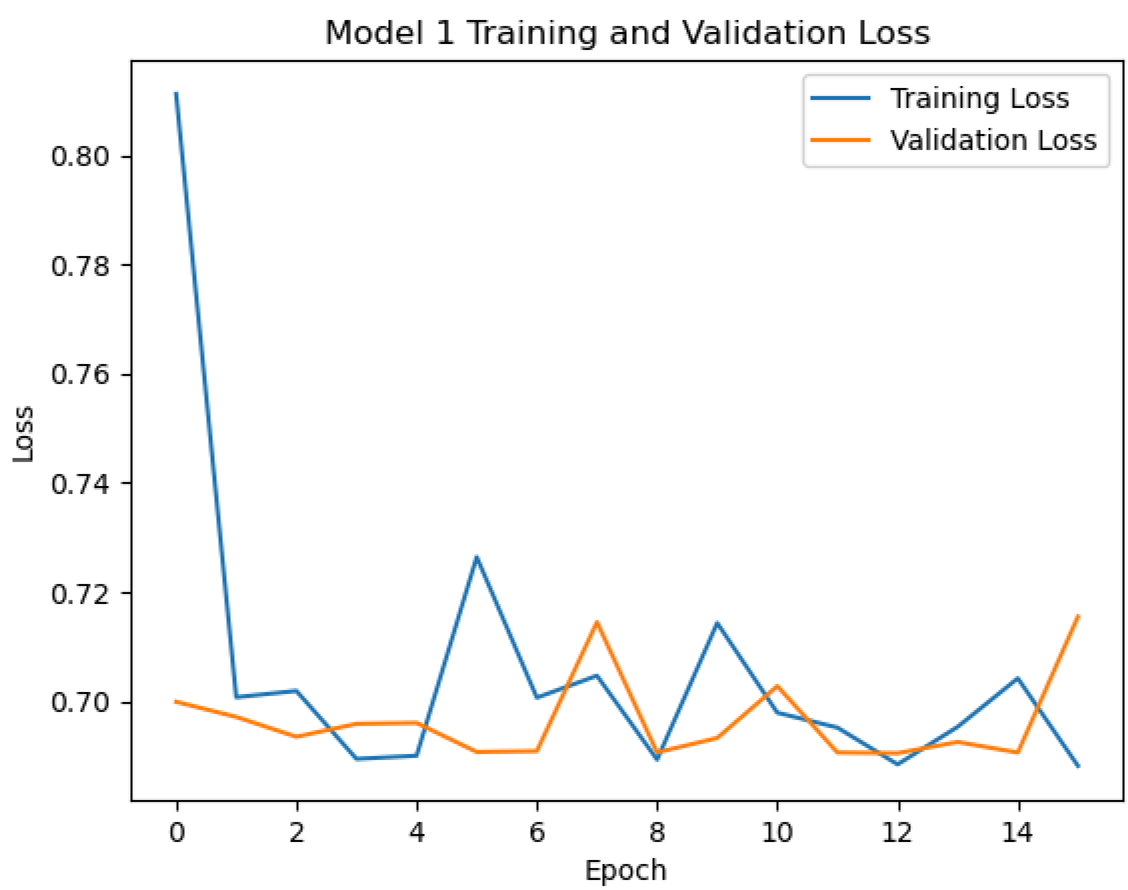
In [58]:
```
import numpy as np
import matplotlib.pyplot as plt

class SimpleNeuralNetwork:
    def __init__(self, input_dim, hidden_dim, output_dim, activation_func
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.activation_function = activation_function
        self.activation_derivative = activation_derivative

        # Initializing the weights and biases
        self.weights1 = np.random.randn(input_dim, hidden_dim)
        self.bias1 = np.zeros((1, hidden_dim))
        self.weights2 = np.random.randn(hidden_dim, hidden_dim)
        self.bias2 = np.zeros((1, hidden_dim))
        self.weights3 = np.random.randn(hidden_dim, output_dim)
        self.bias3 = np.zeros((1, output_dim))

    def forward(self, X):
        # Forward pass
        self.z1 = np.dot(X, self.weights1) + self.bias1
        self.a1 = self.activation_function(self.z1)
        self.z2 = np.dot(self.a1, self.weights2) + self.bias2
        self.a2 = self.activation_function(self.z2)
        self.z3 = np.dot(self.a2, self.weights3) + self.bias3
        exp_scores = np.exp(self.z3)
        self.probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=Tr
        return self.probs

    def backward(self, X, y, learning_rate):
        # Backpropagation
        delta4 = self.probs
        delta4[range(len(X)), y] -= 1
        dW3 = np.dot(self.a2.T, delta4)
        db3 = np.sum(delta4, axis=0, keepdims=True)
        delta3 = np.dot(delta4, self.weights3.T) * self.activation_deriv
        dW2 = np.dot(self.a1.T, delta3)
        db2 = np.sum(delta3, axis=0)
        delta2 = np.dot(delta3, self.weights2.T) * self.activation_deriv
        dW1 = np.dot(X.T, delta2)
        db1 = np.sum(delta2, axis=0)

        # Updating the weights and biases
        self.weights1 -= learning_rate * dW1
        self.bias1 -= learning_rate * db1
        self.weights2 -= learning_rate * dW2
        self.bias2 -= learning_rate * db2
        self.weights3 -= learning_rate * dW3
        self.bias3 -= learning_rate * db3

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

```
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return np.where(x > 0, 1, 0)

def linear(x):
    return x

def linear_derivative(x):
    return np.ones_like(x)

def rmsle(y_true, y_pred):
    return np.sqrt(np.mean(np.square(np.log1p(y_pred) - np.log1p(y_true)

def train(X_train, y_train, X_val, y_val, num_epochs, learning_rate, act
    input_dim = X_train.shape[1]
    output_dim = np.max(y_train) + 1
    hidden_dim = 3

    model = SimpleNeuralNetwork(input_dim, hidden_dim, output_dim, activa

    best_val_loss = float('inf')
    no_improvement_count = 0

    train_losses = []
    val_losses = []

    num_train_samples = X_train.shape[0]

    for epoch in range(num_epochs):
        # Shuffling the training data
        permutation = np.random.permutation(num_train_samples)
        X_train_shuffled = X_train[permutation]
        y_train_shuffled = y_train[permutation]


        for i in range(0, num_train_samples, batch_size):

            X_batch = X_train_shuffled[i:i+batch_size]
            y_batch = y_train_shuffled[i:i+batch_size]

            # Forward pass
            probs = model.forward(X_batch)

            # Compute loss
            corect_logprobs = -np.log(probs[range(len(X_batch)), y_batch
            data_loss = np.sum(corect_logprobs)
            loss = 1./len(X_batch) * data_loss

            # Backpropagation
            model.backward(X_batch, y_batch, learning_rate)

        # Forward pass on validation set
        probs_val = model.forward(X_val)

        # Computing validation loss
```

```
        corect_logprobs_val = -np.log(probs_val[range(len(X_val)), y_val
        val_loss = np.sum(corect_logprobs_val)
        val_loss = 1./len(X_val) * val_loss

        print(f'{model_name} - Epoch {epoch+1}/{num_epochs}, Training Lo

        train_losses.append(loss)
        val_losses.append(val_loss)

        # Early stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            no_improvement_count = 0
        else:
            no_improvement_count += 1
            if no_improvement_count == 3:
                print("Early stopping!")
                break

    # Plotting the training and validation loss
    plt.plot(train_losses, label='Training Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f'{model_name} Training and Validation Loss')
    plt.legend()
    plt.show()

    return model


np.random.seed(0)
X_train = np.random.randn(1000, 2)
y_train = np.random.randint(0, 2, 1000)
X_val = np.random.randn(200, 2)
y_val = np.random.randint(0, 2, 200)


X_test = np.random.randn(200, 2)
y_test = np.random.randint(0, 2, 200)

# Experimenting with hyperparameters
learning_rates = [0.01, 0.001, 0.0001]
num_epochs = 100

activation_functions = [sigmoid, relu, linear]
activation_derivatives = [sigmoid_derivative, relu_derivative, linear_de
model_names = ['Model 1', 'Model 2', 'Model 3']

for i, lr in enumerate(learning_rates):
    print(f"Training with learning rate: {lr}")
    trained_model = train(X_train, y_train, X_val, y_val, num_epochs, lr
    print("Evaluation on test set:")
    test_probs = trained_model.forward(X_test)
    test_predictions = np.argmax(test_probs, axis=1)

    # Calculating RMSLE
```

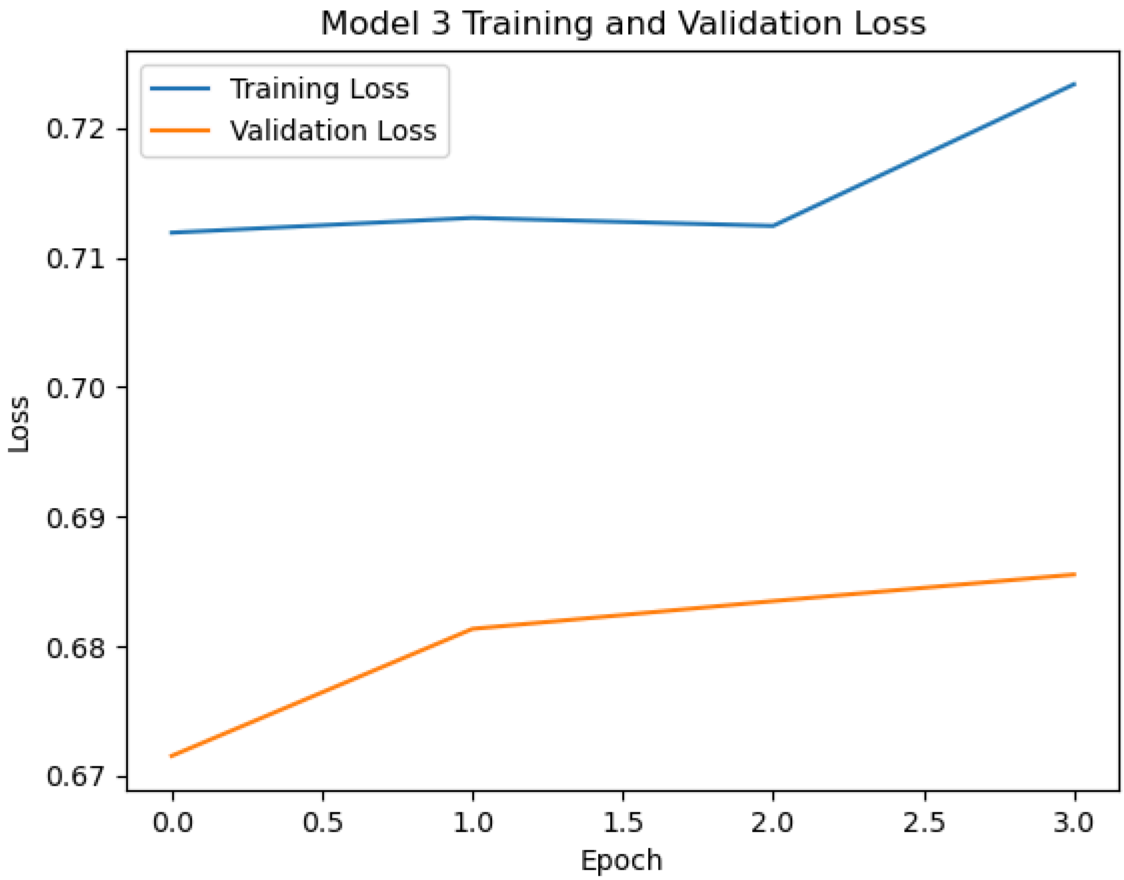
```
    rmsle_value = rmsle(y_test, test_predictions)
    print(f"RMSLE for {model_names[i]}: {rmsle_value:.4f}")
```

```
Training with learning rate: 0.01
Model 1 - Epoch 1/100, Training Loss: 0.8112, Validation Loss: 0.6999
Model 1 - Epoch 2/100, Training Loss: 0.7007, Validation Loss: 0.6971
Model 1 - Epoch 3/100, Training Loss: 0.7019, Validation Loss: 0.6934
Model 1 - Epoch 4/100, Training Loss: 0.6894, Validation Loss: 0.6958
Model 1 - Epoch 5/100, Training Loss: 0.6900, Validation Loss: 0.6960
Model 1 - Epoch 6/100, Training Loss: 0.7264, Validation Loss: 0.6907
Model 1 - Epoch 7/100, Training Loss: 0.7006, Validation Loss: 0.6908
Model 1 - Epoch 8/100, Training Loss: 0.7046, Validation Loss: 0.7145
Model 1 - Epoch 9/100, Training Loss: 0.6893, Validation Loss: 0.6906
Model 1 - Epoch 10/100, Training Loss: 0.7143, Validation Loss: 0.6932
Model 1 - Epoch 11/100, Training Loss: 0.6979, Validation Loss: 0.7027
Model 1 - Epoch 12/100, Training Loss: 0.6951, Validation Loss: 0.6906
Model 1 - Epoch 13/100, Training Loss: 0.6884, Validation Loss: 0.6904
Model 1 - Epoch 14/100, Training Loss: 0.6953, Validation Loss: 0.6925
Model 1 - Epoch 15/100, Training Loss: 0.7042, Validation Loss: 0.6906
Model 1 - Epoch 16/100, Training Loss: 0.6881, Validation Loss: 0.7155
Early stopping!
```



```
Evaluation on test set:
RMSLE for Model 1: 0.4827
Training with learning rate: 0.001
Model 2 - Epoch 1/100, Training Loss: 0.7189, Validation Loss: 0.7185
Model 2 - Epoch 2/100, Training Loss: 0.7103, Validation Loss: 0.6953
Model 2 - Epoch 3/100, Training Loss: 0.6899, Validation Loss: 0.6885
Model 2 - Epoch 4/100, Training Loss: 0.7175, Validation Loss: 0.6915
Model 2 - Epoch 5/100, Training Loss: 0.6863, Validation Loss: 0.6852
Model 2 - Epoch 6/100, Training Loss: 0.6931, Validation Loss: 0.6842
Model 2 - Epoch 7/100, Training Loss: 0.6899, Validation Loss: 0.6896
Model 2 - Epoch 8/100, Training Loss: 0.6897, Validation Loss: 0.6892
Model 2 - Epoch 9/100, Training Loss: 0.6937, Validation Loss: 0.6855
Early stopping!
```


```
Evaluation on test set:
RMSLE for Model 2: 0.5022
Training with learning rate: 0.0001
Model 3 - Epoch 1/100, Training Loss: 0.7119, Validation Loss: 0.6715
Model 3 - Epoch 2/100, Training Loss: 0.7131, Validation Loss: 0.6813
Model 3 - Epoch 3/100, Training Loss: 0.7124, Validation Loss: 0.6834
Model 3 - Epoch 4/100, Training Loss: 0.7234, Validation Loss: 0.6855
Early stopping!
```




```
Evaluation on test set:
RMSLE for Model 3: 0.4877
```