a) The scientist believes that past rainfall and streamflow data (with lags up to 5 days) may be important, but they are unsure which specific lag times are most relevant for each variable. Additionally, they want to avoid using all lagged variables to keep the number of model parameters manageable. If they want to build a model with 5 inputs, how can they determine which lagged variables to include in the model? [Provide a written explanation—no coding or calculations are needed].

In my opinion, I think they could calculate the correlations between the streamflow and the lagged variables. This way will help them to identify which lags have the strongest relationship with streamflow. Also, they could test different combinations of lagged variables through calculate the AIC. And this scientist are familiar with knowledge about hydrological processes, they could determine which lagged variables to include in the model based on their experience.

b) Build a multiple linear regression model to predict streamflow using the following five input variables: rainfall at lag time ($\tau$ = 1, 5) and streamflow at lag times ($\tau$ = 1, 3, 5):

In [97]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
```

In [98]:
```
# Load dataset
df = pd.read_csv('fish_river.csv')
df
```

Out[98]:

| | datetime | year | rainfall | runoff |
|---|---|---|---|---|
| **0** | 2016-01-01 | 2016 | 0.000675 | 1720.0 |
| **1** | 2016-01-02 | 2016 | 0.144395 | 1680.0 |
| **2** | 2016-01-03 | 2016 | 2.918217 | 1640.0 |
| **3** | 2016-01-04 | 2016 | 0.000000 | 1580.0 |
| **4** | 2016-01-05 | 2016 | 0.000000 | 1520.0 |
| **...** | ... | ... | ... | ... |
| **2187** | 2021-12-27 | 2021 | 0.000000 | 989.0 |
| **2188** | 2021-12-28 | 2021 | 3.711825 | 972.0 |
| **2189** | 2021-12-29 | 2021 | 0.018728 | 947.0 |
| **2190** | 2021-12-30 | 2021 | 0.000000 | 928.0 |
| **2191** | 2021-12-31 | 2021 | 0.030013 | 902.0 |

2192 rows × 4 columns

In [99]:

```
# Create lag variables for rainfall and streamflow
for lag in [1, 3, 5]:
    df[f'runoff_lag_{lag}'] = df['runoff'].shift(lag)
for lag in [1, 5]:
    df[f'rainfall_lag_{lag}'] = df['rainfall'].shift(lag)
df = df.dropna()
df
```

Out[99]:

| | datetime | year | rainfall | runoff | runoff_lag_1 | runoff_lag_3 | runoff_lag_5 | ra |
|---|---|---|---|---|---|---|---|---|
| **5** | 2016-01-06 | 2016 | 0.000000 | 1450.0 | 1520.0 | 1640.0 | 1720.0 | |
| **6** | 2016-01-07 | 2016 | 0.000000 | 1390.0 | 1450.0 | 1580.0 | 1680.0 | |
| **7** | 2016-01-08 | 2016 | 0.000000 | 1350.0 | 1390.0 | 1520.0 | 1640.0 | |
| **8** | 2016-01-09 | 2016 | 1.009949 | 1310.0 | 1350.0 | 1450.0 | 1580.0 | |
| **9** | 2016-01-10 | 2016 | 19.440051 | 1290.0 | 1310.0 | 1390.0 | 1520.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **2187** | 2021-12-27 | 2021 | 0.000000 | 989.0 | 1020.0 | 1120.0 | 1120.0 | |
| **2188** | 2021-12-28 | 2021 | 3.711825 | 972.0 | 989.0 | 1070.0 | 1130.0 | |
| **2189** | 2021-12-29 | 2021 | 0.018728 | 947.0 | 972.0 | 1020.0 | 1120.0 | |
| **2190** | 2021-12-30 | 2021 | 0.000000 | 928.0 | 947.0 | 989.0 | 1070.0 | |
| **2191** | 2021-12-31 | 2021 | 0.030013 | 902.0 | 928.0 | 972.0 | 1020.0 | |

2187 rows × 9 columns

In [100…

```
# get the training data and testing data
train_data = df[(df['year'] >= 2016) & (df['year'] <= 2019)]
test_data = df[(df['year'] >= 2020) & (df['year'] <= 2021)]

# define data
X_train = train_data[['runoff_lag_1', 'runoff_lag_3', 'runoff_lag_5', 'rainf
y_train = train_data['runoff']
X_test = test_data[['runoff_lag_1', 'runoff_lag_3', 'runoff_lag_5', 'rainfal
y_test = test_data['runoff']
X_train
```

Out[100…

| | runoff_lag_1 | runoff_lag_3 | runoff_lag_5 | rainfall_lag_1 | rainfall_lag_5 |
|---|---|---|---|---|---|
| **5** | 1520.0 | 1640.0 | 1720.0 | 0.000000 | 0.000675 |
| **6** | 1450.0 | 1580.0 | 1680.0 | 0.000000 | 0.144395 |
| **7** | 1390.0 | 1520.0 | 1640.0 | 0.000000 | 2.918217 |
| **8** | 1350.0 | 1450.0 | 1580.0 | 0.000000 | 0.000000 |
| **9** | 1310.0 | 1390.0 | 1520.0 | 1.009949 | 0.000000 |
| **…** | … | … | … | … | … |
| **1456** | 1630.0 | 1760.0 | 1900.0 | 0.000000 | 3.488331 |
| **1457** | 1570.0 | 1690.0 | 1830.0 | 4.640274 | 0.006127 |
| **1458** | 1510.0 | 1630.0 | 1760.0 | 0.000000 | 0.000000 |
| **1459** | 1450.0 | 1570.0 | 1690.0 | 0.000000 | 0.000000 |
| **1460** | 1400.0 | 1510.0 | 1630.0 | 2.977127 | 0.000000 |

1456 rows × 5 columns

In [101…

```
X_test
```

Out[101…

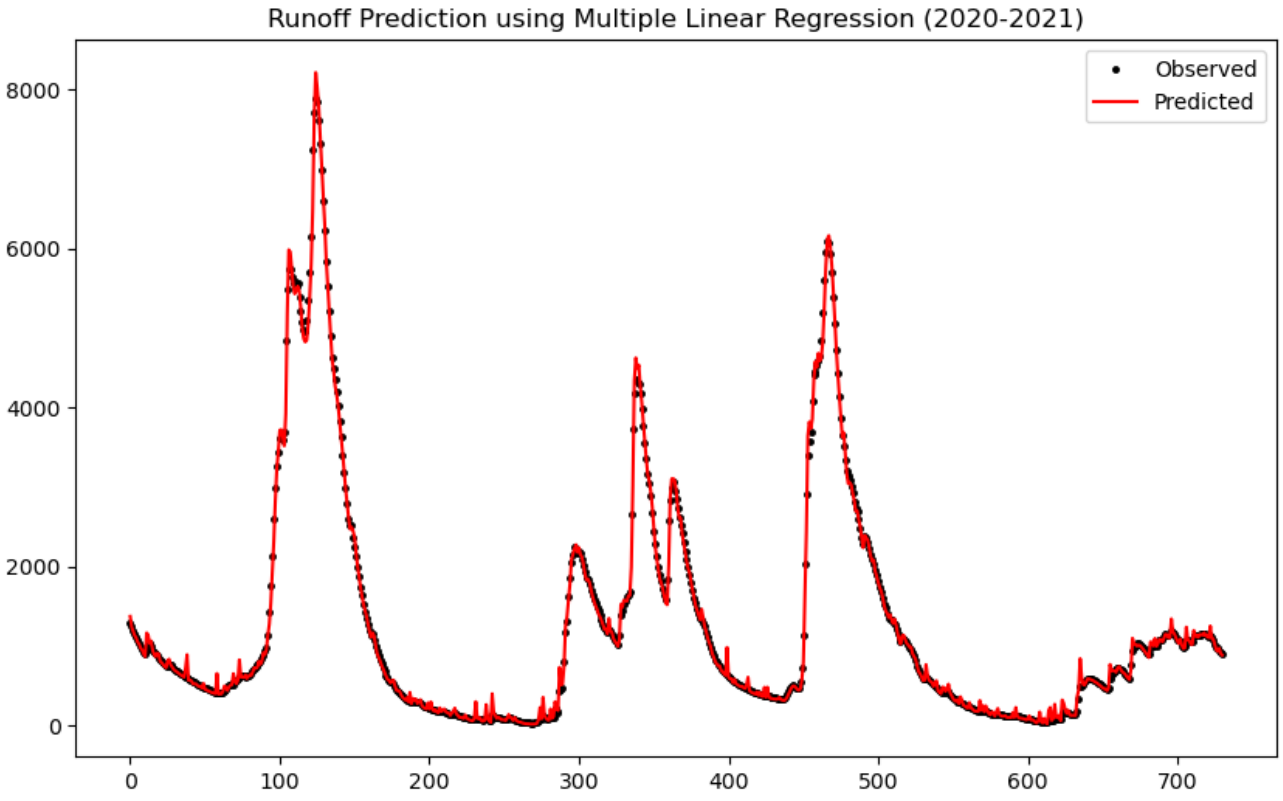
| | runoff_lag_1 | runoff_lag_3 | runoff_lag_5 | rainfall_lag_1 | rainfall_lag_5 |
|---|---|---|---|---|---|
| **1461** | 1350.0 | 1450.0 | 1570.0 | 7.502306 | 4.640274 |
| **1462** | 1300.0 | 1400.0 | 1510.0 | 0.440019 | 0.000000 |
| **1463** | 1250.0 | 1350.0 | 1450.0 | 0.060312 | 0.000000 |
| **1464** | 1200.0 | 1300.0 | 1400.0 | 0.000000 | 2.977127 |
| **1465** | 1160.0 | 1250.0 | 1350.0 | 0.668274 | 7.502306 |
| **…** | … | … | … | … | … |
| **2187** | 1020.0 | 1120.0 | 1120.0 | 0.000000 | 14.028154 |
| **2188** | 989.0 | 1070.0 | 1130.0 | 0.000000 | 0.000000 |
| **2189** | 972.0 | 1020.0 | 1120.0 | 3.711825 | 0.000000 |
| **2190** | 947.0 | 989.0 | 1070.0 | 0.018728 | 0.000000 |
| **2191** | 928.0 | 972.0 | 1020.0 | 0.000000 | 0.000000 |

731 rows × 5 columns

In [102…

```
# Initialize the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# compute the RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f'RMSE: {rmse}')

# compute the Pearson correlation coefficient
pearson_corr, _ = pearsonr(y_test, y_pred)
print(f'Pearson Correlation Coefficient: {pearson_corr}')

# compute the relative bias
relative_bias = (y_pred.mean() - y_test.mean()) / y_test.mean()
print(f'Relative Bias: {relative_bias}')
# Plot the model predictions for the testing period (2020-2021)
plt.figure(figsize=(10, 6))
plt.plot(obs, 'k.', label='Observed', markersize=5)
plt.plot(y_pred, 'r-', label='Predicted')
plt.title('Runoff Prediction using Multiple Linear Regression (2020-2021)')
plt.legend()
plt.show()
```

```
RMSE: 98.10814034558912
Pearson Correlation Coefficient: 0.9979102416958303
Relative Bias: -0.00023792941723045185
```



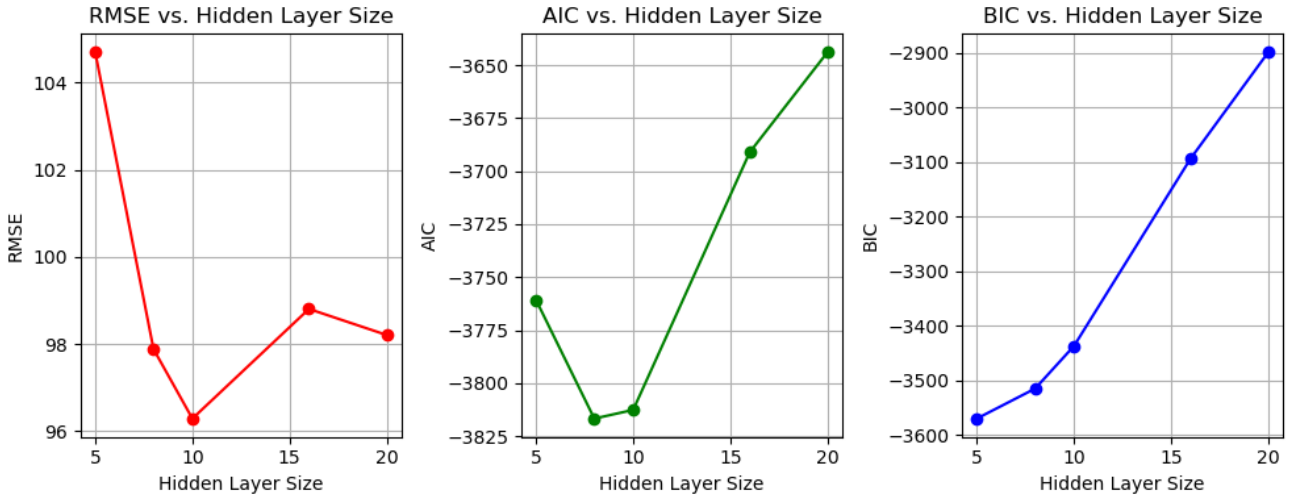
c) Build a Multilayer Perceptron (MLP) model to predict streamflow using the same five input variables in (b) and the same training-testing data split. [Hint: For Python users, utilize the Scikit-Learn MLPRegressor model with default settings unless otherwise specified]

In [103…

```
nodes_list = [5, 8, 10, 16, 20]
results = []

for nodes in nodes_list:
    # # Define the MLP model
    mlp = MLPRegressor(hidden_layer_sizes=(nodes,), max_iter=10000, random_s

    # fit the model
    mlp.fit(X_train, y_train)

    # get the y_predict
    y_pred = mlp.predict(X_test)

    # compute the RMSE
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    # compute Pearson
    pearson_corr, _ = pearsonr(y_test, y_pred)

    # compute the relative bias
    relative_bias = (y_pred.mean() - y_test.mean()) / y_test.mean()

    # compute the number of parameters
    K = (nodes * 5) + (1 * nodes) + nodes + 1

    # compute the AIC and BIC
    n = len(y_train)
    aic = 2 * K + n * np.log(rmse/n)
    bic = K * np.log(n) + n * np.log(rmse/n)

    results.append((nodes, rmse, pearson_corr, relative_bias, aic, bic))

# transform the results to DataFrame
results_df = pd.DataFrame(results, columns=['Nodes', 'RMSE', 'Pearson Correl

print(results_df)

# plot AIC, BIC and RMSE
fig, axs = plt.subplots(1, 3, figsize=(10, 4))

axs[0].plot(results_df['Nodes'], results_df['RMSE'], marker='o', color='red'
axs[0].set_title('RMSE vs. Hidden Layer Size')
axs[0].set_xlabel('Hidden Layer Size')
```

```
axs[0].set_ylabel('RMSE')
axs[0].grid()

axs[1].plot(results_df['Nodes'], results_df['AIC'], marker='o', color='greer
axs[1].set_title('AIC vs. Hidden Layer Size')
axs[1].set_xlabel('Hidden Layer Size')
axs[1].set_ylabel('AIC')
axs[1].grid()

axs[2].plot(results_df['Nodes'], results_df['BIC'], marker='o', color='blue'
axs[2].set_title('BIC vs. Hidden Layer Size')
axs[2].set_xlabel('Hidden Layer Size')
axs[2].set_ylabel('BIC')
axs[2].grid()
plt.tight_layout()
plt.show()

# find the best model
best_aic_model = results_df.loc[results_df['AIC'].idxmin()]
best_bic_model = results_df.loc[results_df['BIC'].idxmin()]
best_rmse_model = results_df.loc[results_df['RMSE'].idxmin()]

print(f'Best model based on AIC: {best_aic_model}')
print(f'Best model based on BIC: {best_bic_model}')
print(f'Best model based on RMSE: {best_rmse_model}')
```

```
   Nodes        RMSE  Pearson Correlation  Relative Bias          AIC  \
0      5  104.696064             0.997651      -0.004867 -3760.755040
1      8   97.884815             0.997916      -0.003246 -3816.700297
2     10   96.291039             0.997981       0.005202 -3812.602235
3     16   98.806248             0.997901      -0.003709 -3691.058442
4     20   98.208742             0.997977       0.012131 -3643.889957

           BIC
0 -3570.550904
1 -3515.543748
2 -3437.477410
3 -3094.028792
4 -2898.923757
```




```
Best model based on AIC: Nodes                    8.000000
RMSE                    97.884815
Pearson Correlation      0.997916
Relative Bias           -0.003246
AIC                 -3816.700297
BIC                 -3515.543748
Name: 1, dtype: float64
Best model based on BIC: Nodes                    5.000000
RMSE                   104.696064
Pearson Correlation      0.997651
Relative Bias           -0.004867
AIC                 -3760.755040
BIC                 -3570.550904
Name: 0, dtype: float64
Best model based on RMSE: Nodes                   10.000000
RMSE                    96.291039
Pearson Correlation      0.997981
Relative Bias            0.005202
AIC                 -3812.602235
BIC                 -3437.477410
Name: 2, dtype: float64
```

Do any of the five MLP models outperform the multiple-linear regression model? If so, why might that be the case?

Yes. MLP can capture nonlinear relationships between the input and output variables better than MLR. So if the relationship is nonlinear, MLP may provide better predictions. And MLP especially with more nodes and layers, have a higher capacity to learn complex patterns from the data. However, if an MLP model is too complex relative to the size of the training dataset, it may overfit the training data. Also, if the dataset is small, linear regression might perform better.

Identify the best MLP model based on each metric. Do all metrics agree? If not, explain

why?

No, not all metrics agree. This is because the RMSE foucus on minimizing error, while AIC and BIC foucus on model complexity. So based on AIC and BIC, we could choose the simpler model which RMSE helps us to choose more accuracy model.

The Maine Department of Environmental Protection is particularly interested in developing a data-driven model that excels at predicting extreme runoff values (e.g., the maximum daily runoff in a year or daily runoff values above the 95th percentile). Discuss two approaches that could be used to enhance the model's performance in predicting extreme events, even if this comes at the expense of the model's accuracy in predicting non-extreme runoff values. [Hint: Consider the choice of loss function and regularization techniques.]

1. Loss Functions

Traditional loss functions such as the mean square error (MSE) tend to focus on minimizing the overall error of all predictions, which can lead to poor performance when predicting extreme values. In order to improve the prediction of extreme runoff values, greater weights can be assigned to the extreme values. For example, a weighted loss function could assign a greater penalty to errors in runoff values above the 95th percentile, ensuring that the model prioritizes accurate prediction of high runoff events. Another approach is to use quantile regression, which predicts a specific percentile (e.g., the maximum daily runoff in a year or daily runoff values above the 95th percentile). The quantile loss function minimizes the error for a given quantile, ensuring that the model is appropriate for predicting extreme high runoff events rather than focusing on the median or mean. This helps improve the performance of extreme runoff prediction.

2. regularization techniques

L1 regularization (Lasso) expresses the sparsity of the model, which means that only the most important features for predicting extreme values are retained. Regularization based on Extreme Value Theory (EVT) is specifically designed to improve the prediction of rare and extreme events. By introducing EVT, models can emphasize the tail behavior of the runoff distribution and improve the model's ability to accurately predict large runoff values.